משה ליכטמן עמית רש

המדריך השלם לשפת C

רב מכר

מהדורה שישית

תוכנת TCLite וקוד מקור

הוצאת הוד-עמי
ספרי מחשבים

המדריך השלם

לשפת C

מהדורה שישית

יש להתעלם מכל מה שקשור לתקליטור מצורף

קבצי המקור נמצאים באתר הוד-עמי

www.hod-ami.co.il

בקטגוריה "קבצי תרגול לספרים" יש לבחור

ב- 59159 כדי להוריד קובץ 59159.zip

מוקדש באהבה ל**נורית**, **יעל** ומשפחותינו

ולזכרו של אבי, **יונה ליכטמן** ז"ל

עורך ראשי: **יצחק עמיהוד**

עריכה מקצועית לעדכון המהדורה: **אבי בוץ**

עריכה לשונית ועיצוב: **שרה עמיהוד**

עיצוב עטיפה: **סטודיו מצגר**

שמות מסחריים

שמות המוצרים והשירותים המוזכרים בספר הינם שמות מסחריים רשומים של החברות שלהם. הוצאת הוד-עמי עשתה כמיטב יכולתה למסור מידע אודות השמות המסחריים המוזכרים בספר זה ולציין את שמות החברות, המוצרים והשירותים. שמות מסחריים רשומים (registered trademarks) המוזכרים בספר צוינו בהתאמה.

Borland Turbo C/C++ - מוצר של חברת Borland

MS-DOS, Windows - מוצרים של חברת Microsoft

הודעה

ספר זה מיועד לתת מידע אודות מוצרים שונים. נעשו מאמצים רבים לגרום לכך שהספר יהיה שלם ואמין ככל שניתן, אך אין משתמעת מכך כל אחריות שהיא.

המידע ניתן "כמות שהוא" ("as is"). הוצאת הוד-עמי אינה אחראית כלפי יחיד או ארגון עבור כל אובדן או נזק אשר ייגרם, אם ייגרם, מהמידע שבספר זה, או מהתקליטור שמצורף לו.

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

- **טלפון: 09-9564716**
- **פקס: 09-9571582**
- **דואר אלקטרוני: info@hod-ami.co.il**
- **אתר באינטרנט: http://www.hod-ami.co.il**

המדריך השלם

לשפת C

מהדורה שישית

עמית רש
משה ליכטמן

הוצאת הוד-עמי
ספרי מחשבים

The complete Guide to the C language

6th Ed.

By

Amit Resh

Moshe Lichtman

Editor: **I. Amihud**

(C)

כל הזכויות שמורות

הוצאת הוד-עמי

לספרי מחשבים בע"מ

ת.ד. 6108 הרצליה 46160

טלפון: 09-9564716   פקס: 09-9571582

אין להעתיק או לשדר בכל אמצעי שהוא ספר זה או קטעים ממנו בשום צורה ובשום אמצעי אלקטרוני או מכני, לרבות צילום והקלטה, אמצעי אחסון והפצת מידע, ללא אישור בכתב מאת ההוצאה, אלא לשם ציטוט קטעים קצרים בציון שם המקור.

הודפס בישראל

מהדורה 6 - 2000, 2001

All Rights Reserved

HOD-AMI Ltd.

P.O.B. 6108, Herzliya

ISRAEL, 2000, 2001

ISBN 965-361-128-3 מסת"ב

תוכן עניינים מקוצר

תוכן העניינים

פרק 1 אבני הבניין של השפה 47

פרק 2 משפט ההצבה ומבנה התוכנית 65

פרק 3 קלט ופלט באמצעות פונקציות ספריה 77

פרק 4 משפטי בקרה בשפת C 87

פרק 5 פונקציות 111

פרק 6 מערכים ומחרוזות 129

פרק 9 אפיון משתנים וטווח ההכרה 183

פרק 10 הגדרות קדם-מעבד 195

פרק 11 מצביעים 203

פרק 12 מבנים 233

נספח ב' נהלים לכתיבת תוכנית ב-C 424

נספח ג' פונקציות העכבר 437

נספח ד' תקציר פקודות ופונקציות (Quick Reference)..... 453

נספח ה' התקליטור המצורף וסביבת העבודה TCLite.......... 469

קורא יקר, שים לב!

התוכניות הורצו בהצלחה בתוכנה המצורפת TCLite, וגם במהדרים שלמים.

לעיתים תמצא הבדל בין אותה תוכנית הנמצאת בתקליטור לבין התוכנית בספר. דבר זה נובע מכך שהתוכנית בספר מוצגת בחלקים לצורך ההסבר.

"הספריה הסטנדרטית" אינה כה סטנדרטית ולכן, למשל, בעוד שבמהדר השלם פונקציות מסוימות (כדוגמת ()getch) נמצאת בספריה stdio, במהדר TCLite היא נמצאת בספריה conio. לפיכך, צריך לכתוב גם את הספריה conio בפקודה include#.

אם תמצא שגיאה בתוכנית כלשהי נשמח לשמוע על כך ולדאוג לתיקונה במהדורה הבאה.

הקדמה

שפת C פותחה בראשית שנות ה-70 על ידי דניס ריצ׳י (Dennis M. Ritchie) ממעבדות Bell שבארה״ב. שמה ניתן לה עקב היותה ״צאצא״ ישיר של שפת התכנות B, שנכתבה בשנת 1970 על ידי קן תומפסון (Ken Thompson) עבור מחשב PDP-7.

לשפת C קשר הדוק עם מערכת ההפעלה UNIX, שעבורה פותחה השפה לראשונה. ואמנם, מערכת ההפעלה UNIX ותוכנות העזר שלה נכתבו רובן בשפת C. במקורה נועדה שפת C לפיתוח מערכות הפעלה ומהדרים אחרים, אולם במשך הזמן היא נמצאה מתאימה לכתיבת תוכניות בתחומים רבים אחרים, הן בעיבוד נתונים והן במערכות הנדסיות ויישומי בקרה וזמן-אמת. שפת C רכשה לעצמה אוהדים רבים תוך מספר שנים מועט, וכיום היא אחת השפות הנפוצות ברוב מערכות המחשב הקיימות.

שפת C הינה שפה קלה יחסית ללימוד ונוחה לתכנות. השפה מאפשרת פקודות בקרה מובנות ונוחות לשימוש, ביטויים מתומצתים ושימוש באופרטורים רבים בכמה רמות היררכיה.

אחד המאפיינים הבולטים של שפת C הוא הצמצום שלה. השפה מכילה מעט פקודות, מספר קטן של טיפוסי משתנים ואינה כוללת פקודות קלט/פלט. תכונות אלו, הנראות במבט ראשון כחיסרון, הן אשר מקנות לשפה יתרונות רבים. בזכותן, המהדר של השפה קטן ופשוט, תוצאות ההידור לפקודות בסיסיות יעיל ביותר, כושר הניידות של השפה בין מערכות מחשב שונות ומערכות הפעלה רב, וניתן ללמוד אותה במהירות. את החסר בשפה משלימות ספריות המערכת הצמודות למהדר, אשר עליהן נרחיב את הדיבור במהלך הספר. הן מיועדות לספק שירותי קלט/פלט, טיפול במבני נתונים, ניהול הזיכרון ועוד.

בתחילה הוגדרה שפת C באופן לא רשמי על ידי יוצריה במעבדות Bell, ולמעשה היתה מיועדת לשימוש פנימי בלבד. לאחר מספר שנים, כאשר זכתה לפופולריות רבה, היא זכתה להכרה ולהגדרת תקן על ידי **ANSI** (**A**merican **N**ational **S**tandard **I**nstitute). אגב כך גם הוכנסו בה מספר הרחבות לעומת תצורתה המקורית. כיום, רוב המהדרים הקיימים לשפה מותאמים לתקן זה בדרגת תאימות גבוהה. על-כן, ניתן להעביר תוכנית הכתובה ב-C ממערכת מחשב אחת לאחרת, כמעט ללא שינויים.

שפת C מופעלת בתחנות עבודה ומיני-מחשבים של יצרנים שונים בפיקוחן של גרסאות שונות של UNIX ומערכות הפעלה אחרות המגדירות עצמן כ-UNIX (XENIX, AIX, VENIX ועוד). שפת C גם מופעלת במחשבים מרכזיים שונים ובמערכות הפעלה שאינן ממשפחת UNIX, כדוגמת מערכת VM/SP של יבמ.

מהדרי C מופעלים במחשבים אישיים תואמי יבמ תחת מערכות ההפעלה Windows MS-DOS ו-XENIX. הפופולריות הרבה של שפת C ושל מחשבי IBM-PC ותואמיהם תרמה לכך שכיום קיימים בשוק עשרות מהדרי C שונים למחשבים אישיים.

המהדרים הבולטים בשוק הם אלה של Microsoft ושל Borland ועליהם מבוסס ספר זה, ובעיקר על מהדר Borland C/C++ הנמצא בשימוש ברוב בתי התוכנה בארץ.

בתחילה, היו לשפת C אוהדים ומשתמשים רבים במוסדות להשכלה גבוהה בלבד, בהם השתמשו במערכת ההפעלה UNIX. אחר-כך הצטרפו למעגל המשתמשים בשפה מהנדסים ואנשי מדע אחרים בתחומים שונים ובעיקר מתחומי האלקטרוניקה, הבקרה ומערכות זמן אמת. בתי תוכנה גילו את היתרונות הרבים של השימוש ב-C ואימצו אותה כשפת תכנות עיקרית ליישומים שונים. התפוצה הרבה של המחשב האישי ותחנות העבודה האישיות והימצאותם של מהדרי C רבים למחשבים אלה תרמה להרחבת מעגל המשתמשים בשפה רבת עוצמה זו.

בעשור האחרון המשיכה שפת C להתפתח בכיוון של תכנות מונחה עצמים (Object Oriented) לאחר מאמץ שהוביל מדען בשם ביורן סטרוסטרופ ממעבדות AT&T (חברת הטלפונים האמריקאית המכילה את מעבדות Bell).

הגירסה של שפת C שפיתח סטרוסטרופ קיבלה את השם **C++**. הכוונה מאחורי השם שנבחר היא שפת C שקודמה לשלב הבא. פשר הבחירה בצירוף ++ יובן במהלך קריאת הספר, כאשר האופרטור ++ בשפת C יוסבר.

הפופולריות של שפת C++ קיבלה דחיפה חזקה כאשר חברת Microsoft אימצה אותה כשפת הפיתוח המרכזית של מערכת ההפעלה Windows ושל יישומים הנכתבים עבור Windows.

קיים ויכוח בעולם בין אנשי מחשבים לגבי היתרונות והחסרונות של השימוש בשפת C++ לעומת שפת C המקורית, עבור סוגי יישומים שונים. כיום מקובל לחשוב שיש יתרון ברור לשימוש ב- C++ עבור יישומי Windows ואינטרנט ואילו שפת C עדיפה ביישומי זמן אמת ומערכות משובצות מחשב. אולם הגבולות אינם חדים ובשנים האחרונות רואים נטיה גוברת לפיתוח בשת C++, מכיון ששפה זו מבוססת על שפת C ולמעשה מהווה הרחבה שלה, על ידי תוספת של כלים התומכים בגישה מונחית עצמים. מומלץ מאוד לאנשי תוכנה המתכננים ללמוד C++, להתחיל בלימוד שפת C. שליטה טובה בשפת C, המהווה את הבסיס לשפת C++, תאפשר מעבר קל ונוח יותר ללימוד C++ בהמשך.

ספר זה יצא לאור לראשונה בשנת 1988 ומאז זכה לחמש מהדורות עדכון ולאלפי תלמידים מרוצים בכל רחבי הארץ. הוא גם נקבע כספר לימוד במוסדות חינוך רבים. לפניך **המהדורה השישית המורחבת**, אשר מבוססת על נסיונם הרב והמקיף של המחברים, העורכים ושל מורים רבים אשר הואילו לתרום מנסיונם בעבודה עם ספר זה. לכולם, התודה.

הספר מיועד לכל המשתמשים הקיימים והצפויים, במטרה לתת להם כלי עזר ללימוד השפה ולשיפור רמת המיומנות בהפעלתה. יופיה של השפה ועוצמת שירותיה למשתמש הינם אתגר אינטלקטואלי, אשר ראוי להתמודד איתו. תמורתו - תכנות יעיל ואלגנטי בשפת C.

הנחיות לקורא

ספר זה מכוון ללימוד מקיף של שפת C, אשר **מתאים לקורא בעברית**, לדרך החשיבה שלו ולצורת עבודתו. במהלך הלימוד מודגשת העבודה עם טקסטים בעברית, אשר כידוע, שונה מאוד משפה לטינית ולכן מעוררת קשיים בשילובה בשפת תכנות.

הספר מיועד לתלמידי תיכון, סטודנטים, חובבי מחשב ותכנות ולמהנדסים המעוניינים ללמוד שפת תכנות, אשר תשרת אותם בעבודתם. למתחילים בתכנות, אנו ממליצים לקרוא בעיון את פרק המבוא, כדי שתכנות בכלל ושפת C לא יהיו זרים לו.

בספר כלולים דוגמאות תכנות רבות, תרשימים, הסברים מפורטים ותרגילים בכל פרק (כפי הצורך). הם מכוונים להדגיש את כל התכונות והאפשרויות של השפה ואת טכניקות התכנות האפשריות והמקובלות בה. פרק 18, משימות, מציג בפני הקורא אתגרי תכנות בדרגות קושי שונות, המותאמות לכל אחד מפרקי הספר. מומלץ לנסות ולפתור אותם תחילה, לפני שבוחנים את הפתרון.

דוגמאות התכנות המובאות בספר, הפתרונות לתרגילים (לא כולם!) ופתרונות המשימות **הורצו** במהדר Borland Turbo C/C++ Lite (נמסר באדיבות **ברק שירותי תוכנה**). התוכניות, פתרונות התרגילים והמשימות ניתנים **בתקליטור המצורף** (ראה להלן). ניתן ללמוד את השפה ולפתור את התרגילים גם באמצעות מהדרים אחרים הקיימים בשוק, ואשר עומדים לרשות הקורא.

מבנה הספר

הספר מכיל 18 פרקים, חמישה נספחים ואינדקס כפול, בעברית ובאנגלית.

מבוא לתכנות בשפת C "טבילת אש". סקירת מושגי תכנות בסיסיים לטובת מתכנתים מתחילים, והדרכה לקריאת תוכנית בשפת C. בסיום הלימוד הקורא יוכל "לקרוא" תוכנית פשוטה בשפת C והיא לא תהיה זרה לו.

פרק 1 תיאור אבני הבניין של השפה: המשתנים, הקבועים והאופרטורים והפעולות שניתן לבצע בעזרתם.

פרק 2 הצגת משפטי התכנות ומבנה תוכנית בשפת C. תיאור משפט ההצבה המאפשר לשנות ערכי-משתנים.

פרק 3 הסבר תמציתי על אפשרויות פעולות קלט/פלט באמצעות פונקציות ספריה סטנדרטיות. ב**פרק 16** ניתן הסבר מקיף בנושא קלט/פלט. הסבר ראשוני זה

מכוון לאפשר ביצוע פעולות קלט/פלט בסיסיות בדוגמאות ובתרגילים שבפרקים הבאים.

פרק 4 משפטי הבקרה השונים בשפת C. באמצעות משפטים אלה ניתן להשפיע על מהלך התוכנית: לבצע התניות, לולאות והסתעפויות.

פרק 5 תיאור הפונקציה, הגדרתה, השימוש בה והדרכים להעביר אליה פרמטרים ולקבל תוצאות. ניתנות דוגמאות לפונקציות שונות, השימוש בהן והסבר על הספריה הסטנדרטית.

פרק 6 מבני מערכים שונים ומחרוזות. ניתן הסבר על אופן ההגדרה של מערכים ומחרוזות, אתחולם ודרכי השימוש בהם. בפרק זה גם הסבר על שיטות מיון שונות.

פרק 7 פרק חשוב זה מדריך את הלומד לעצב את התוכנית בדרך נכונה שתקל עליו את התחזוקה בעתיד. על הקניית הרגלי תכנות טובים ראה גם **נספח ב'**.

פרק 8 לאחר כתיבת תוכניות פשוטות מגיעים לשלב בו בונים תוכנית שלמה ממספר קובצי מקור ופונקציות ספריה - זוהי בניית "פרויקט".

פרק 9 טווח ההכרה של משתנים בפונקציות ובקטעי תכנות שונים הינו אחד מהנושאים החשובים בתכנות. בפרק זה מוסבר אפיון המשתנים ואתחולם.

פרק 10 תיאור משפטי הבקרה למהדר, המאפשרים ביצוע של הידור מותנה.

פרק 11 מציג את המצביעים, המהווים ציר מרכזי בעוצמתה של שפת C. מוסברים בהרחבה מטרת השימוש במצביעים ואופן הגדרתם, שימושים שונים במצביעים ודרכים לפתור בעיות תכנות בפונקציות ובמערכים.

פרק 12 הלומד יכיר את הדרכים ליישום מבני-נתונים מורכבים. הפרק דן באופן הגדרת המבנה, אתחולו וצורת השימוש בו, וגם מציג שימושים בסוגי מבנים שונים.

פרק 13 האיגוד הוא מבנה נתונים משלים. פרק זה מתעכב על שימוש נפוץ באיגוד, כאשר הוא משולב בתוך מבנה.

פרק 14 תיאור של מספר שימושים מתקדמים בשפת C: מצביעים למסך, השלמות בנושאי חיפוש ומיון, הקצאת זיכרון ועבודה בסיביות.

פרק 15 רקורסיה היא טכניקת תכנות בעלת עוצמה רבה, אשר נתמכת היטב בשפת C. הפרק מציג שיטות לתכנון פונקציות רקורסיביות ועומד על היתרונות והאפשרויות הרבות הגלומות בהן.

פרק 16 תיאור מפורט של פונקציות קלט/פלט ובעיקר - פונקציות של ספריות המערכת שנועדו לשימוש בקבצים. הפרק מסביר טכניקות תכנות שונות, עם מאגד ובלעדיו ועוסק גם בניתוב מחדש.

פרק 17 שפת C מאפשרת עבודה ברמת החומרה שמשמעותה, בין השאר, גישה לפסיקות המערכת ועיבודה ברמת ה-BIOS. בנושאים אלה דן הפרק.

פרק 18 בפרק זה רוכזו משימות התכנות, בחלוקה לפי פרקי הספר (פרט לפרקים הראשונים). בכל משימה מוגדרת הבעיה שיש לפתור, או התוכנית לביצוע, והיא מלווה בדברי הסבר ולעיתים המלצה לדרך הפתרון. רוב המשימות פתורות ומוצגות כתוכניות מוכנות בתקליטור המצורף לספר.

נספח א׳ בנספח זה מוצג תיאור כללי של מבנה המחשב. מוסברים היחידות הפנימיות במעבד ועקרונות פעולתם, מבלי להיכנס להסברים מפורטים המצריכים ידע באלקטרוניקה. הקורא יוכל למצוא בנספח זה הסבר מהותי למושגים בסיסיים במבנה המחשב כגון: מעבד, זיכרון, אוגרים, אפיק, פסיקות ועוד. ניתן הסבר על שיטות ייצוג נתונים ואריתמטיקה בבסיס בינארי ובבסיס הקסדצימלי, וגם על תקני ASCII.

נספח זה מיועד לרענן מושגים מסוימים במבנה המחשב, אשר התייחסנו אליהם במהלך הלימוד של השפה ולטובת אלה שמתכנתים ברמת החומרה.

נספח ב׳ כתיבת תוכנית נחשבה בעבר אומנות ויש עדיין הסבורים כך. אולם עם זאת, אפשר להקנות לתכנות הרגלי עבודה נכונים כדי שהתוכנית לא תישאר נחלתו של הכותב אותה בלבד, אלא תוכל להיות קריאה ומובנת לו בעתיד, וגם לאחרים שעשויים להיות מעורבים בתחזוקה שלה במועד אחר. התיעוד הוא אחד מאבני הנגף החמורים בתחזוקת תוכנה, ונספח זה מיועד לתרום לשיפור.

נספח ג׳ העכבר הינו אחד מהתקני המחשב החיוניים. שליטה בו וניצול יכולתו הינם דרישת קדם בתוכניות רבות. קבוצת הפונקציות בנספח זה מכסה את רוב הפעולות הדרושות.

נספח ד׳ הקורא מקבל בתקציר זה רשימה מרוכזת של ההגדרות והפקודות בשפת C. יש בו גם תיאור קצר של פונקציות הספריה החשובות והפקודות המיוחדות של **קדם המעבד** (pre processor), בהן משתמשים במהלך הלימוד בספר.

נספח ה׳ הסבר **מפורט** על תוכן התקליטור ועל סביבת העבודה **TCLite**. **אל תפעיל את התקליטור** בטרם קראת בעיון את ההוראות. כך תחסוך מעצמך עוגמת נפש וטרחה מיותרת בעת שיבושים שעלולים להיות.

אינדקס האינדקס מוצג בשני מיונים: עברית ואנגלית, כדי להקל על הלומד והמעיין בספר.

הערות כלליות

במהלך הספר נקטנו במספר כללים כדי לפשט הסברים ודוגמאות:

א. ברוב תוכניות הדוגמה קבענו "**מספרי שורות**" לשורות הקוד, כדי שניתן יהיה להתייחס לשורה מסוימת או לקבוצת שורות בהסבר הניתן בהמשך. מספור זה **אינו** קיים בתוכנית המעשית וגם **אסור** לכתוב אותו, כי אינו חלק מהשפה.

ב. בהגדרה פורמלית של פקודות ומשפטי בקרה השתמשנו בסימנים < > כדי לתחום מרכיב מסוים בפקודה. תוכן הסימנים מסביר את המשמעות של אותו מרכיב.

ג. בחלק מהדוגמאות נאלצנו לפצל שורה מסוימת בגלל מגבלת רוחב הדף. פיצול זה אינו חלק מהשפה ואינו הכרחי בתוכנית מעשית. אדרבא, רצוי לכתוב את הפקודה בשורה רצופה. עם זאת, אנו ממליצים לכתוב שורות קוד קצרות, כדי להקל עלינו את הכתיבה, ההבנה והשליטה בתוכנית.

תוכניות, תרגילים ומשימות והתקליטור המצורף

התרגילים שבסוף פרקי הלימוד מייצגים קשת רחבה של בעיות, ולפתרונם יש להשתמש במשפטים ובטכניקות שנלמדו באותו פרק. אין לנו ספק, שפתרון תרגילים אלה ישמש נדבך נוסף של ניסיון הנרכש בשפת C שאפשר יהיה לנצל אותו לאחר מכן גם בעבודה מעשית.

במהלך הלימוד יש הפנייה ל**משימות** מתקדמות המפורטות בפרק 18. במשימות אלו מוגדרות בעיות שפתרונן דורש כתיבת תוכנית מחשב מורכבת יותר מזו שנדרשת בתרגילים השוטפים. לעיתים ניתן גם הסבר מפורט עבור הלומד, הכולל רקע והצעות לשיטת הפתרון. **הדגשנו את השימוש בשפה העברית**, כדי להדריך את הקורא לקראת עבודה מעשית. אנו ממליצים להתמודד עם משימות אלו במהלך לימוד הספר, שכן הדרך הטובה ביותר ללמוד שפת תכנות, היא באמצעות תרגול התכנות וההרצה במחשב.

לרוב התרגילים והמשימות הכנו פתרונות, המצויים ב**תקליטור** המצורף, כדי לסייע ללומד ולחלץ אותו אם נתקל בקושי כלשהו. אנו ממליצים בפניך הקורא, לנסות ולהתמודד עם הבעיה בטרם תעיין בפתרון המצורף. בסיום תוכל להריץ את הפתרון המוצע, להשוות אותו עם שלך ולראות היבט נוסף לפתרון של בעיה נתונה.

יש משימות המסתמכות על התוצאות של משימות קודמות. במקרה כזה תוכל להיעזר בפתרונות המצורפים, באם לא הצלחת לפתור משימה בשלמותה.

הסבר מפורט להתקנת התקליטור תמצא בנספח ה', בקובץ ONCD שבספריית השורש בתקליטור. יש לקרוא אותם בעיון לפני כל פעולה, כדי לא לגרום נזק.
בתקליטור נמצא מהדר התרגול TCLite (Borland Turbo C/C++ Lite) (הוראות התקנה והפעלה בנספח ה' ובקובץ TCLiteGuide.pdf שבתקליטור בספריה software\tclite), את רוב התוכניות וקטעי התכנות המופיעים בספר, פתרונות לתרגילים שבכל פרק (לא לכולם), פתרונות לחלק מהמשימות הנתונות בפרק 18 ומספר קובצי נתונים הנחוצים להרצת כמה מן התוכניות.

שים לב: קטעי התכנות משובצים בתוך תוכנית ראשית, כדי שיהיה ניתן להריץ אותם במחשב. הקורא יוכל לבצע שינויים כדי לבחון דרכים נוספות לתכנות או הרצה.

שים לב: בתקליטור **לא** כלולות כל התוכניות **ואין** פתרונות לכל התרגילים או לכל המשימות, אלא כפי שמצאנו לנכון להציג לקורא.

התוכניות שבתקליטור הורצו במהדר TCLite (Borland Turbo C/C++ Lite) שנמצא בתקליטור. אפשר, וגם רצוי להריץ אותם במהדר מלא של השפה. ראוי להזכיר שלפעמים יהיה צורך לעשות מספר שינויים בתוכנית או בפקודות קדם-המהדר (כמו למשל הפקודה include), כדי לאפשר הרצה ללא שגיאות.

מבוא לתכנות בשפת C

טבילת האש

לפני שנתחיל בלימוד שפת C, נסביר מושגים, נסקור בקצרה את מרכיביה ונציג מספר דוגמאות פשוטות. בפרקים הבאים נדון בהרחבה בכל נושא מאלה המפורטים כאן ובנושאים רבים אחרים.

- התכנות בשפת C כולל שימוש רב בפונקציות ובמודולריות גבוהה.
- שפת C משמשת היום בכל תחומי התכנות בענף המחשבים.
- ב-C אין טיפול ישיר בקלט/פלט, קובצי גרפיקה או כל תחום אחר. השימוש נעשה בעזרת פונקציות ספריה.
- **פונקציית ספריה** - קטע תוכנית, שמישהו כתב עבורנו לטיפול בנושאים אלה.
- בשפת C ניתן לטפל במיגוון סוגי נתונים.
- מספר שלם הוא משתנה מסוג integer.
- משתנה הוא תא בזיכרון, או מספר תאים, בהתאם לסוג משתנה. לכל סוג נדרש מספר תאים שונה, לכל משתנה ניתן שם על-פי בחירתנו.
- משתנה, על-פי שמו, פירושו שהערך אשר נמצא בו (בתא הזיכרון) יכול להשתנות. הדבר תלוי בתוכניתן בלבד, והשינוי מתבצע על ידי השמת ערך או תוצאות חישוב למשתנה.
- ערך של כל משתנה יכול להיות בטווח מספרים מסוים. משתנה integer נמצא בטווח 32,767 עד 32,768-.

מושגים בסיסיים

1.	**מהדר** (Compiler)	תוכנית המתרגמת תוכנית מקור משפת תכנות כלשהי (שפת C לדוגמה) אל שפת המכונה המתאימה למחשב.
2.	**שפת מכונה** (machine language)	שפה אשר המחשב מבין ורק הוא (למעשה, סוג או משפחת המחשבים). לכל משפחת מחשבים שפת מכונה משלה.
3.	**משפט** (statement)	פקודה בשפת התכנות.
4.	**בלוק** (block)	קבוצת פקודות המתבצעת כיחידה לוגית אחת.
5.	**משתנה** (Variable)	תווית המייצגת ערך לא קבוע אשר נמצא בזיכרון. ערך זה עשוי להשתנות במהלך התוכנית.
6.	**קלט** (Input)	תהליך הזנת נתונים למחשב לשם עיבוד.
7.	**פלט** (Output)	תהליך הצגת הנתונים המתקבלים לאחר ביצוע הקלט.
8.	**אתחול** (initialization)	הצבת ערך התחלתי במשתנה, לפני תחילת השימוש בו בתוכנית.
9.	**ספריה** (library)	אוסף תוכניות ופונקציות המאוחסנות במחשב ועומדות לרשותן של תוכניות, אשר נכתוב בשפת C.

כיצד נכתוב תוכנית בשפת C

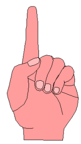
ראשית, עלינו לדעת מה רוצים מאיתנו. זאת אומרת, לקרוא את הדרישה ולהבינה, לפרק דרישה זו לתת-דרישות, ולענות על כל תת-דרישה בנפרד.

דרישה לדוגמה: קלוט שני מספרים, חשב והדפס את הסכומים.

תת-דרישה

- ביצוע פעולות קלט - לצורך זה נשתמש בפונקציות ספריה מתאימות. פונקציות הספריה המטפלות בקלט/פלט יושבות בספריית stdio. לשם כך נשתמש בפונקציה scanf לקלט ובפונקציה printf לפלט.
- יש להגדיר תאי זיכרון כדי לאחסן את מספרי הקלט. נחליט על שמות לתאי זיכרון אלה. השמות הם a ו-b. וההגדרה היא:

```
int a,b;
```



שים לב!

נקודה-פסיק *(;) הוא תו מוסכם המציין סיום משפט קוד.*

❖ יש להגדיר תא זיכרון נוסף לצורך אחסנת תוצאות החיבור בין שני המספרים. נחליט ששמו של תא הזיכרון הוא sum ונוסיף אותו להגדרה הקודמת.

```
int a, b, sum;
```

❖ לאחר שהכנו את הרקע, יש לעבור לחלק הביצועי. תחילה עלינו לקלוט שני משתנים, וההגדרה היא:

```
scanf("%d", &a);
scanf("%d", &b);
```

❖ פעולת החישוב מבוטאת בשורת הקוד הבאה:

```
sum = a + b;
```

❖ מה שנותר הוא להציג את התוצאה, וזו פעולת הפלט.

```
printf("%d", sum);
```

❖ התוכנית הבאה נותנת מענה לדרישה: קלוט שני מספרים, חשב והדפס את סכומם.

```
/* chp0001.c  :קובץ תקליטור  */

/* הכרזה על ספריות בשימוש על מנת להשתמש בפונקציות מתוך ספריות אלו */
/* הספריה stdio מטפלת בפעולות קלט/פלט */
/* פונקציות קלט (scanf) פלט (printf) נמצאות בספריה זו */

#include <stdio.h>

/*main הפונקציה הראשית של התוכנית , פונקציה זו חייבת להימצא בכל תוכנית C */

void main(void)
  {
   int a, b, sum;      /* הכרזה על משתנים פרטיים של הפונקציה main */
                       /* החלק הביצועי של התוכנית */
        /* אין אפשרות להכריז על משתנים נוספים אחרי תחילת החלק הביצועי */
   printf("הקש מספר ראשון   :");       /* הצג משפט זה על המסך (פלט) */
   scanf("%d", &a);                     /* בקש ערכים למשתנה a (קלט) */
   printf("הקש מספר שני   :");          /* הצג משפט זה על המסך (פלט) */
   scanf("%d", &b);                     /* בקש ערכים למשתנה b (קלט) */

   sum = a + b;
            /* חבר את ערכם של a ו- b והצב את התוצאה במשתנה sum */
                       /* הצג את תוצאת החישוב על המסך */
  printf("%d הוא %d -ו %d הסכום של", sum ,a, b);
  }
```

תוצאה

```
הקש מספר ראשון:  4
הקש מספר שני:     6
הסכום של 6 ו-4 הוא 10
```

הסבר בסיסי של פעולות קלט/פלט

הפונקציה Printf (פלט)

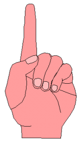
- הפונקציה מאפשרת פלט על-פי תבנית של מלל וערכים של תאים בזיכרון.
- הפונקציה תשבץ בפלט את הערכים בכל מקום שיסומן "**%d**" במחרוזת הפלט.

דוגמה:

```
printf("The sum of %d and %d is %d\n", a, b, a+b);
```

נניח ש-a ו-b קיבלו את הערכים 4 ו-6 בהתאמה.

אז יודפס: `The sum of 4 and 6 is 10`

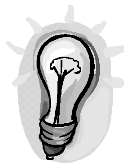


שים לב!

הסימון \n במחרוזת הוא סימון מיוחד הגורם לירידת שורה במסך.

הפונקציה Scanf (קלט)

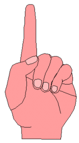
- הפונקציה מאפשרת קלט על-פי תבנית של תאים בזיכרון.



טיפ!

לא ניתן להדפיס הודעות למשתמש באמצעות scanf, אלא רק באמצעות printf.

- הפונקציה תקלוט מספרים לפי מספר הפעמים שבו נמצא הצירוף %d במחרוזת תבנית הקלט.



שים לב!

הפונקציה scanf משנה תאי זיכרון שאינם שלה. כדי לאפשר זאת, יש להוסיף את התו "&" לפני שם המשתנה.

תרגילים

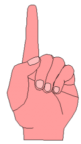
1. קלוט שלושה מספרים והדפס את מכפלתם.
2. קלוט מספר והדפס את הריבוע שלו.

תנאים

משפט if משמש אותנו להגדיר בתוכנית תנאי ביצוע.

דוגמת תחביר א'

```
if (תנאי)
      משפט
else
      משפט
```

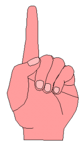


שים לב!

- ***משפט*** *(statement) הוא* ***ביטוי*** *(שורת קוד).*
- *משפט else אינו חובה.*
- *כל עוד התנאי "**אמת**" (true), יבוצע המשפט (ביטוי) - אחרת, כלומר "**שקר**" (false) יבוצע המשפט שלאחר else.*
- *תנאי "אמת" הוא כאשר ערך תנאי* ***שונה*** *מ-0.*

דוגמת תחביר ב'

```
if (תנאי)
   {
          בלוק
   }
else
   {
          בלוק
   }
```



שים לב!

- *בלוק הוא ביטוי מורכב (סדרה של שורות קוד).*
- *כאשר רוצים להתייחס לסדרה של שורות קוד, יש לסמן סדרה זו בין סוגריים מסולסלים ({ }), אחרת יתבצע רק המשפט הראשון שלאחר התנאי, בהנחה שהתנאי אמת.*

דוגמה CHP0002.C

קלוט שני מספרים ובדוק אם הראשון גדול יותר מהשני, או אם שניהם שווים, והצג הודעה מתאימה.

```
                /* chp0002.c :קובץ תקליטור */

#include <stdio.h>

void main(void)
 {
  int a, b;

  printf("הקלד מספר ראשון : \n");
  scanf("%d", &a);
  printf("הקלד מספר שני : \n");
  scanf("%d", &b);
  if (a > b)
     printf("המספר הראשון גדול יותר\n");
  else
     if (b > a)
          printf("המספר השני גדול יותר\n");
     else
          printf("המספרים שווים\n");
 }
```

תוצאות הרצה אפשריות

1

```
הקלד מספר ראשון:
5
הקלד מספר שני:
8
המספר השני גדול יותר
```

2

```
הקלד מספר ראשון:
8
הקלד מספר שני:
4
המספר הראשון גדול יותר
```

3

```
הקלד מספר ראשון:
5
הקלד מספר שני:
5
המספרים שווים
```

תרגיל

3. קלוט שלושה מספרים והדפס את הגדול מביניהם.

לולאות

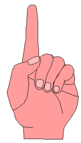
הפקודה while

מבנה תחבירי א'

```
while    (תנאי)
           משפט
```

מבנה תחבירי ב'

```
while    (תנאי)
      {
           בלוק
      }
```



שים לב!

- *כל עוד התשובה לתנאי היא **כן (אמת)** - הלולאה תמשיך להתבצע.*
- *יש לדאוג לכך שגם תהייה יציאה מהלולאה. זאת אומרת, לדאוג שתוצאת תנאי הלולאה תהיה **שקר**. אחת הדרכים לעשות זאת היא עם מונה לולאה.*

הגדרת מונה לולאה ואיפוסו:	`int i=0;`
כל עוד מונה הלולאה קטן מ-10	`while (i < 10)`
	`{`
הדפס את ערכו של מונה הלולאה	`printf("%d", i);`
קדם את המונה ב-1	`i=i+1;`
	`}`

כאשר מונה הלולאה שווה ל-10, אנו מסיימים את הלולאה.

דוגמה CHP0003.C

הדפס 10 פעמים את המילה "שלום".

```
                    /* chp0003.c  :קובץ תקליטור  */

#include <stdio.h>

void main(void)
 {
   int i;                        /*     הכרזה על משתנה - מונה לולאה  */

   i = 0;                        /*                איפוס מונה הלולאה  */
   while (i < 10)         /*  כל עוד התנאי נכון (אמת) הלולאה תתבצע  */
    {
```

```
    printf("שלום\n");
    i = i + 1;      /* קידום מונה הלולאה */
                    /* כאשר מונה הלולאה יגיע ל- 10 הלולאה תסתיים */
    }
}
```

תוצאת הריצה של התוכנית

```
שלום
שלום
שלום
שלום
שלום
שלום
שלום
שלום
שלום
שלום
```

תרגילים

4. קלוט 5 מספרים והדפס את סכומם.
5. הדפס את סכום המספרים מ-1 ועד 10.

תנאים מורכבים

תנאי מורכב נבנה מתנאים פשוטים יותר.

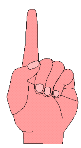
הפעולות המרכיבות תנאים מורכבים הן: "וגם", "או", "לא".

סימנים מוסכמים

&& - **וגם**

דוגמת תחביר:

```
if (תנאי1 && תנאי2)
        משפט
```



שים לב!

משפט if יתבצע רק כאשר תנאי 1 יהיה אמת וגם תנאי 2 יהיה אמת.

|| - או

דוגמת תחביר:

```
if (תנאי1 || תנאי2)
      משפט
```

שים לב!

משפט if יתבצע רק כאשר תנאי 1 יהיה אמת או תנאי 2 יהיה אמת.

! - לא

דוגמת תחביר:

```
if (! (תנאי))
      משפט
```

שים לב!

*משפט if יתבצע רק כאשר התנאי יהיה שקר. המשפט **שקר** (אינו נכון) מקביל למשפט **לא מתקיים**.*

דוגמה CHP0004.C

קלוט מספרים חיובים וסכם אותם. יש לקלוט לכל היותר 5 מספרים, או עד שייקלט מספר שלילי. הדפס את סכום המספרים.

```
                 /* קובץ תקליטור: chp0004.c */

#include <stdio.h>

void main(void)
 {
  int sum = 0, i = 0, input = 0;   /* הגדרת משתנים ואתחולם */

  while (i < 5 && input >= 0)
   {
     scanf("%d", &input);
     if (input >= 0)
         sum = sum + input;
     i = i + 1;
   }
  printf("sum = %d", sum);
 }
```

תוצאות הרצה אפשריות

```
1
2
3
4
5
sum = 15
```

1

```
3
4
3
-1
sum = 10
```

2

תרגיל

6. קלוט מספרים חיוביים או שליליים ומצא את המקסימלי מביניהם. יש לקלוט עד שייקלט המספר 0 או 5 מספרים לכל היותר.

משפט for

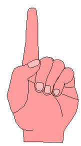
לולאת for נועדה ללולאות עם מונה בקרה.

דוגמת תחביר א'

```
for (אתחול; תנאי המשך; פקודת קידום)
          משפט
```

דוגמת תחביר ב'

```
for (אתחול; תנאי המשך; פקודת קידום)
{
          בלוק
}
```



שים לב!

אתחול *- מתבצע פעם אחת בלבד בכניסה ל-for.*

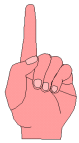
תנאי המשך *- לולאת for תתבצע כל עוד תנאי ההמשך מתקיים.*

פקודת קידום *- נועדה לקידום מונה הלולאה.*

תהליך ביצוע for

```
        0        1           3
for (אתחול; תנאי המשך; פקודת קידום)

             משפט/בלוק
                2
```



שים לב!

תחילה יבוצע האתחול (0), לאחר מכן תתבצע בדיקת נכונותו של התנאי (1). סדרת הפקודות משפט/בלוק (2) תתבצע לאחר שווידאנו שהתנאי אמת, ולבסוף יקודם מונה הלולאה (3). התהליך חוזר על עצמו ((1) (2) (3)) כל עוד התנאי מתקיים.

ניתן להשוות לולאה זו עם לולאת while ומונה.

```
i=o;
while (i<10)
   {
         משפט/בלוק
      i=i+1;
   }
```

דוגמה CHP0005.C

הדפס 10 פעמים את המילה "שלום".

```
                 /* chp0005.c :קובץ תקליטור */
#include <stdio.h>
void main(void)
 {
  int i;

  for (i = 0; i < 10; i = i + 1)
     printf("שלום\n");
 }
```

תוצאות הרצת התוכנית

```
שלום
שלום
שלום
שלום
שלום
שלום
שלום
שלום
שלום
שלום
```

תרגיל

7. כתוב תוכנית המדפיסה את לוח הכפל.

פונקציות

פונקציה היא היחידה הבסיסית המרכיבה תוכנית ב-C. היא למעשה תוכנית קטנה הממלאת משימה מוגדרת אחת.

הגדרה של פונקציה

```
סוג הערך המוחזר     שם הפונקציה     (רשימת פרמטרים)
{
        גוף הפונקציה
}
```

שם הפונקציה

שם הפונקציה יכול להיות מורכב מתווים גדולים או קטנים ומקו תחתון "_" להפרדה בין מילים. שם הפונקציה צריך להיות בעל משמעות אשר תרמוז על מהות הפונקציה. למשל פונקציה המחברת בין שני משתנים, שמה צריך להיות, למשל add, add_numbers, או AddNumbers.

רשימת פרמטרים

רשימת המשתנים, אשר מועברים לפונקציה (משוכפלים בזיכרון כעותקים), ונועדו לצורכי חישוב פנימיים של הפונקציה.

גוף הפונקציה

שורות קוד אשר מהוות את יחידת הביצוע של הפונקציה.

סוג הערך המוחזר

תוצאת הביצוע של הפונקציה, כמו למשל, פונקציה אשר מחזירה תוצאת חיבור בין שני מספרים מסוג שלם. כאשר תוצאת החיבור היא מספר מסוג שלם (זה איננו חוק, אלא החלטה של התוכניתן או הדרישה), זאת אומרת שהערך המוחזר הוא מסוג שלם.

תכונות

- כל פונקציה יכולה לקבל פרמטרים מהפונקציה שקראה לה ולהחזיר ערך.
- כל פונקציה יכולה לקרוא לפונקציות אחרות.
- לפונקציה יש משתנים מקומיים שהם פרטיים לה. זאת אומרת - רק היא יכולה לשנות את ערכם.
- משתנים מקומיים קיימים כל עוד הפונקציה קיימת (למשך זמן פעולת הפונקציה).
- עם סיום הפונקציה משתחררים תאי הזיכרון של המשתנים המקומיים והם פנויים לצרכים אחרים.
- הפונקציה main היא הפונקציה הראשית בתוכניות בשפת C וחייבת להימצא בכל תוכנית. פונקציה זו נקראת על ידי מערכת ההפעלה ואליה היא מחזירה ערך (אין חובה להחזיר ערך מהפונקציה הראשית).
- עיקרון התכנות בשפת C מבוסס על רמה היררכית של פונקציות, כאשר כל פונקציה עונה לתת-בעיה בסיסית יותר.

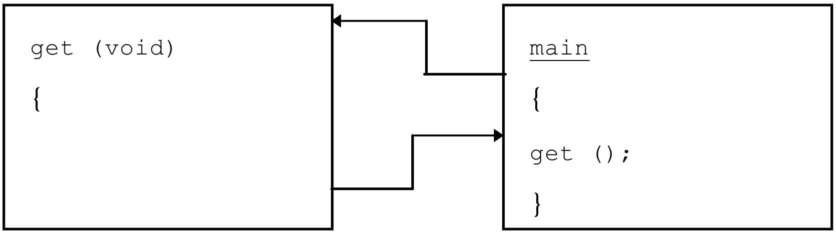
סוגים של פונקציות

פונקציה קולטת מקבלת נתונים מן המשתמש.

דוגמה

```
void get(void)
{
   int a;

   scanf("%d", &a);
   }
```

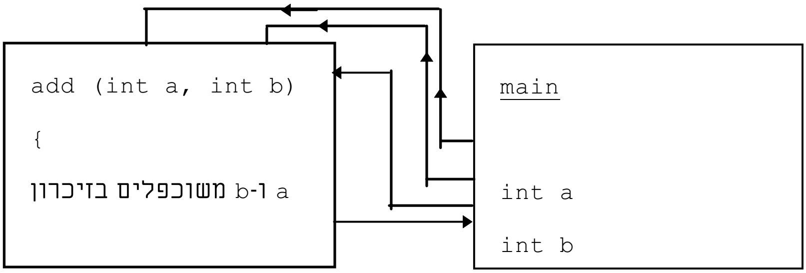


תרשים 1

פונקציה מקבלת הפרמטרים מועברים לפונקציה בעת הפעלתה.

דוגמה:

```
void add (int a, int b)
{
   printf ("%d", a+b);
}
```



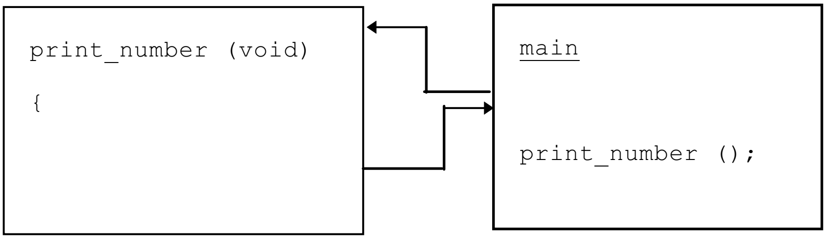
תרשים 2

פונקציה מדפיסה פונקציה המבצעת פלט.

דוגמה:

```
void print_number (void)
{
   int num;

   num = 8;
   printf ("%d", num);
}
```



תרשים 3

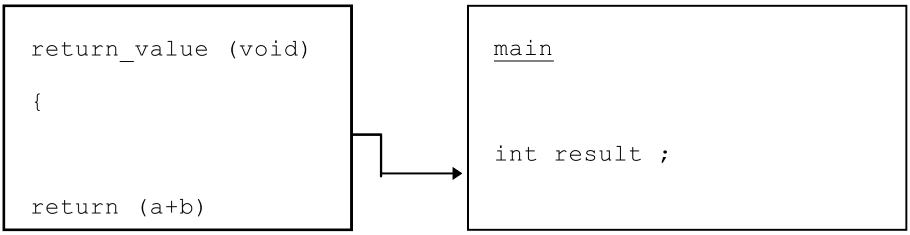
פונקציה מחזירה מחזירה ערך לפונקציה שהפעילה באמצעות הפקודה return.

דוגמה:

```
void return_value (void)
{
   int a, b;

   a=6;
   b=8;

   return (a+b);
}
```



תרשים 4

דוגמה CHP0006.C

כתוב פונקציה המקבלת מספר חיובי שלם, ומחזירה את סכום המספרים מ-1 ועד לאותו מספר.

```
                /* chp0006.c :קובץ תקליטור */

#include <stdio.h>

int add_to_num(int num);                 /* הכרזה על הפונקציה */

void main(void)
 {
  int num;                  /* המשתנה שעד אליו נחשב את כל המספרים */
  int result;                       /* המשתנה שיקבל את התוצאה */

  printf("הקש מספר : ");
  scanf("%d", &num);
  result = add_to_num(num); /* קריאה לפונקציה (הפעלת הפונקציה) */
  printf("\n תוצאה = %d", result);
 }

/*
                                        ערך מוחזר: ערך מסוג integer
                                         שם הפונקציה: add_to_num
                                         פרמטר מועבר: num מסוג : int
*/

int add_to_num(int num)                        /* הגדרת הפונקציה */
 {
  int sum;                 /* משתנה מקומי אשר יאחסן את תוצאת החישוב */
  int i;          /* משתנה אשר משמש אותנו כמונה מספרים עד למספר num
*/

  sum = 0;
  for (i = 0; i <= num; i = i + 1)
     sum = sum + i;
  return(sum);     /* sum הוא הערך המוחזר ומאוחסן במשתנה result
*/
 }
```

דוגמת הרצה

```
הקש מספר : 4

תוצאה = 10
```

תרגילים

8. כתוב פונקציה בשם max המקבלת שני מספרים ומחזירה את הערך המקסימלי מביניהם.
9. כתוב פונקציה בשם triangle הקולטת מספר ומדפיסה משולש כוכביות בגודל המספר. לדוגמה: למספר 4.

```
   *
  * *
 * * *
* * * *
```

תחילה עלינו לכתוב פונקציה בשם star_line המקבלת מספר ומדפיסה שורת כוכבים באורך המספר. על הפונקציה triangle להשתמש בפונקציה star_line.

קלט/פלט של תווים

תאי זיכרון מסוג int משמשים ב-C לאחסון מספרים שלמים. תא זיכרון מסוג char משמש לאחסון מספרים קטנים (127 ÷ 128-), ובדרך-כלל משמש לאחסון קוד ASCII של תווים (ראה נספח טבלת ASCII).

יש פונקציות ספריה המיועדות לטיפול בקלט/פלט של תווים:

putchar( ); מקבלת כפרמטר קוד של תו ומדפיסה אותו על המסך.

getchar( ); מחזירה קוד של תו שנקלט מקובץ הקלט של התוכנית. כאשר קובץ הקלט מסתיים, תחזיר הפונקציה 1-.

קובץ קלט התוכנית ממתינה להקלדת תווים על המסך. סדרת התווים לאחר ההקלדה נקראת קובץ קלט.

סיום קלט סיום הקלט יתבצע עד להקשת F6 או Ctrl+Z.

הערה לפונקציה getchar () יש שני חלקים. חלק אחד מאפשר קליטת תווים ואגירתם; החלק השני מאפשר למשוך את התווים החוצה בעזרת לולאה, כדי להציג אותם.

דוגמה:

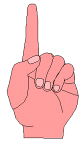
```
int ch ;
ch = getchar ();
while (ch! = -1)
{
   ch = getchar ();
}
```

יש לציין שאנו יכולים לקבוע כל תו רצוני לציון סיום קובץ הקלט.

לדוגמה:

```
int ch ;
while ((ch = getchar()) !='\n');
   putchar (ch + 1);
```

במצב זה נקליד סדרת תווים עד להקשת Enter (למעבר שורה).



שים לב!

- *שורת הקוד ;(putchar (ch + 1 תציג את התו הבא על המסך. זאת אומרת, אם הקלדנו a , יוצג b על המסך.*
- *קוד ASCII של a הוא 97 כאשר נוסיף לו 1, התו שמייצג 98 הוא b (ראה טבלת ASCII).*

דוגמה CHP0007.C

כתוב תוכנית הסופרת את מספר השורות בקובץ הקלט.

```
                 /* chp0007.c :קובץ תקליטור */
#include <stdio.h>
void main(void)
 {
  int lines;                    /* משתנה הסופר את השורות בקובץ הקלט */
  int ch;              /*משתנה אשר מקבל את התווים בקובץ הקלט בזה אחר זה*/
  char eof = -1;     /*משתנה המייצג את קוד סוף הקובץ המתקבל אוטומטית*/

   lines = 0;
   printf("Enter your file F6 or Ctrl+z to exit : \n");
   while (( ch = getchar() ) != eof)
                                /* קרא תווים עד שמתקבל תו סוף הקובץ */
      if (ch == '\n') lines = lines + 1;    /* ספור את השורות */
  printf("The number of lines in your file is : %d\n", lines);
 }
```

תוצאות הרצת התוכנית

```
Enter your file F6 or Ctrl+Z to exit:
avi
boots
^z
The number of lines in your file is: 2
```

ניתן לקלוט ולהציג תווים גם בעזרת פונקציית הספריה אשר למדנו. לדוגמה:

```
char ch ;
scanf("%c", fch ); קלט
printf("%c", ch);    פלט
```

תרגילים

10. כתוב פונקציה בשם toupper המקבלת קוד של תו ואם הוא אות אנגלית קטנה, הפונקציה תחזיר את הקוד של האות האנגלית הגדולה המתאימה, אחרת - הפונקציה תחזיר את הערך שנלקח כפרמטר.

(רמז: ההפרש בין אות אנגלית קטנה לאות אנגלית גדולה בטבלת ASCII הוא 32).

11. כתוב תוכנית המדפיסה את קובץ הקלט שלה מוסב לאותיות גדולות.

רמז: getch() פונקציה הקולטת תו.

getche( ) פונקציה הקולטת תו ומציגה אותו על המסך.

דוגמה:

```
char ch ;
ch = getch();
```

משתנים בוליאניים

תוצאת ביטוי תנאי בשפת C הוא מספר המיוצג על ידי הערך אפס כאשר הוא שקר, וכל ערך השונה מ-0 כאשר הוא אמת.

אופרטורים בוליאניים הם:

&& **וגם**

|| **או**

! **היפוך ("לא").**

בעזרתם ניתן לקבל תשובה מספרית (אמת או שקר, true או false בהתאמה) לגבי ערכם של ביטויים.

דוגמה:

```
int a, b, c;

a=2
b=a || 0            /*  b יקבל את הערך אמת     */
c = !b              /*  C יקבל את הערך שקר     */
if (c)              /*  אם c בעל הערך אמת      */
  printf("True");   /*  הדפס "True"           */
else                /*  אחרת                  */
  printf("False")   /*  הדפס "False"          */
```

תוצאה

בסיום הרצת קטע התוכנית יודפס "False", מפני ש-c קיבל את ערכו ההפוך של הערך b, שהוא ערך אמת. כלומר, c יהיה בעל ערך שקר.

תרגילים

12. כתוב פונקציה בשם isupper המקבלת קוד של תו, ואם הוא אות אנגלית גדולה, היא מחזירה אמת.

13. כתוב פונקציה בשם islower המקבלת קוד של תו, ואם היא אות אנגלית קטנה, היא מחזירה אמת.

פרק 1

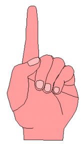
אבני הבניין של השפה

בכל שפה, ושפת מחשב בכלל זה, דרושות אבני בניין כדי להרכיב את הביטויים והמשפטים שלה, ובמקרה שלנו - את תוכנית המחשב. בפרק זה נכיר את הקבוצות העיקריות המהוות בסיס לשפת מחשב בכלל, ולשפת C בפרט: המשתנים (מטיפוסים שונים), הקבועים והפעולות (אופרטורים) האפשריות ביניהם.

טיפוסים של משתנים

טיפוס (Type) של משתנה או קבוע נקבע על פי ההגדרה של המתכנת. גם לפונקציות אשר נכיר בהמשך, נקבע טיפוס על פי הטיפוס של הערך המוחזר מהן. המהדר קובע טיפוס לביטויים האריתמטיים המופיעים בתוכנית, על פי סדרת חוקים המופעלים על האיברים בכל ביטוי. בהמשך נדון בחוקים אלה.

בטבלה הבאה מוצגים הטיפוסים המוכרים בשפת C והגודל של כל משתנה מאותו טיפוס. גדלים אלה אופייניים לרוב המהדרים.



שים לב!

*גודל כל טיפוס צוין ביחידות של **בתים** (בית הינו תא זיכרון בן 8 סיביות).*

טיפוס	הגדרה	גודל	תחום	
תו	char	בית	127	-128
שלם	int	2 בתים	32,767	-32768
ממשי	float	4 בתים	$3.4x10^{38}$	$3.4x10^{-38}$
ממשי כפול	double	8 בתים	$1.7x10^{308}$	$1.7x10^{-308}$
ריק (לא מוגדר)	void	לא מוגדר	לא מוגדר	

שים לב!

*בתוכניות C **שלמים** משמשים בחישובים אריתמטיים, בלולאות, באינדקסים, במערכים ולמטרות נוספות. **תווים** משמשים בעבודה עם **מחרוזות** (Strings), לקלט של נתונים מהמשתמש (במקלדת) ועוד.*

כאשר בתוכנית נכללים חישובים מתמטיים במספרים **ממשיים** - משתמשים בטיפוסים **float** ו-**double**. ההבדל ביניהם, פרט לגודלם, הוא ברמת הדיוק. מספר ממשי מורכב משלושה חלקים: **מנטיסה** (Mantissa), **מעריך** (Exponent) **וסימן**. ערכו של המספר הממשי מחושב לפי המכפלה: מעריך^2 * מנטיסה * סימן.

בטיפוס double, הערך המקסימלי של המנטיסה ושל המעריך גדול מאשר בטיפוס float, ולכן דיוקו וטווח המספרים שהוא מייצג רב יותר. במרבית היישומים משתמשים בטיפוס float. לדוגמה, במחשב מסוג IBM-PC הטיפוס float הוא בן 4 בתים וטיפוס double כפול ממנו, בן 8 בתים.

שים לב!

- *א. תחום המספרים מטיפוס שלם הוא: 32,767 <-- 32,768-. טעות נפוצה נובעת מהתעלמות מעובדה זו, והצבת ביטוי שערכו חורג מתחום זה לתוך משתנה שלם.*
- *ב. בשפת C לא קיים טיפוס לוגי **בוליאני** (Boolean). זהו טיפוס המקבל שני ערכים בלבד: **אמת - True ושקר - False**. לכן נהוג בשפת C, שכל ביטוי שערכו **שונה מאפס** הוא אמת ורק ביטוי שערכו אפס הוא שקר.*

דוגמה לביטויים שערכם אמת:

```
322
(15-6)
```

דוגמה לביטויים שערכם שקר:

```
0
(5-5)
```

הגדרה משנית של טיפוסים

כדי להשלים את המנגנון, המוגבל מעט, של הגדרת הטיפוסים, קיים בשפת C מנגנון **להגדרה משלימה** (Modifiers), המהווה הרחבה לארבעת הטיפוסים שהכרנו. הגדרות אלו מיועדות למספרים שלמים בלבד.

הגדרה	גודל	תחום
short	2 בתים	-32768 → + 32767
long	4 בתים	-2,147,483,648 → + 2,147,483,647
unsigned int	2 בתים	0 → 65,535
unsigned char	בית	0 → 255
unsigned long	4 בתים	0 → 4,294,967,295

מנגנון ההגדרה של טיפוסי משתנים

מנגנון ההגדרה של טיפוסי משתנים בשפת C מאפשר לנו להגדיר **טיפוס חדש** באמצעות שילוב של טיפוסים קיימים. מנגנון ההגדרה נקרא **Typedef**.

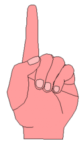
ההגדרה מתבצעת באופן הבא:

typedef unsigned char byte;

byte הוא טיפוס חדש שהגדרנו (שם שאנו בחרנו) מסוג unsigned char, אשר גודלו 8 סיביות.

typedef unsigned int twobytes;

twobytes הוא טיפוס חדש שהגדרנו (שם שאנו בחרנו) מסוג unsigned int, אשר גודלו 16 סיביות.



שים לב!

רצוי מאוד ששם הטיפוס החדש יעיד על אופיו.

פעולת ההגדרה נקראת - **Data type definition**.

התוכנית הבאה ממחישה את פעולת מנגנון ההגדרה:

```
#include <stdio.h>
#include <conio.h>
     /* data type definition - הגדר טיפוס נתונים חדש */
typedef unsigned char byte;
void main (void)
{
byte ch;                /*בעזרת הטיפוס החדש אנו מגדירים משתנים חדשים*/
clrscr();               /*נקה את המסך*/
ch = getch();           /*קלוט תו*/
printf("%c", ch);       /*הדפס תו*/
}
```

א. כפי שציינו קודם, תחום המספרים בטיפוס שלם אינו קבוע, ותלוי בסוג המהדר והמחשב בו משתמשים. כאשר דרוש יישום שבו הייצוג של מספר שלם הוא בדיוק שני בתים, רצוי להשתמש בטיפוס short. טיפוס זה מוכר על ידי כל המהדרים וגודלו קבוע.

ב. כאשר הביטויים בקטע תוכנית מסוים הם **חיוביים בלבד**, וברצונך לבצע חישובים במספרים גדולים, ניתן להשתמש בטיפוסים **חסרי סימן** (Unsigned). במקרה זה התחום החיובי של הטיפוס גדל פי שניים.

ג. ביישומי **זמן-אמת** (Real Time) ו/או כאשר הזיכרון הנתון לרשותך הוא מוגבל - מומלץ להשתמש עבור מספרים קטנים (למשל עבור מונים) בטיפוס unsigned char. במקרה כזה נדרש עבור המשתנה בית בודד, לתחום המספרים מ-0 עד 255.

הגדרת משתנים

הגדרת **המשתנים** בשפת C נעשית בצורה הבאה:

<טיפוס המשתנה> <משתנה ראשון>, <משתנה שני>, ... ;

יש לדאוג לרווח של תו אחד לפחות בין טיפוס המשתנה לרשימת המשתנים. אנו ממליצים לעצב את המבנה של התוכנית, כדי שתהיה קריאה יותר, ולהשתמש לשם כך בטבלר (טבולטור).

לדוגמה, הגדרות של משתנים:

משתנה מטיפוס שלם	`int i ;`
משתנה מטיפוס תו	`char ch ;`
משתנה שלם גדול, עם סימן	`long big_num ;`
משתנים ממשיים רגילים	`float fl_1, fl_2 ;`

שם המשתנה חייב להתחיל באות, דהיינו: A-Z או a-z. ניתן להתחיל שם משתנה **בקו תחתון** "_" (Underscore), אך מומלץ לא לעשות זאת, שכן מרבית המהדרים משתמשים בתו זה עבור המשתנים הפנימיים שלהם. שאר התווים בשם המשתנה יכולים להיות אותיות, מספרים או קו תחתון. אין להשתמש **בתו רווח** ' ' (blank) בתוך שם משתנה. כדי ליצור הפרדה בין מילים בשם, יש להשתמש בקו תחתון. דוגמה למשתנים חוקיים:

```
float result_1;
int number, Count;
```

דוגמה למשתנים לא חוקיים:

(מתחיל בספרה)	`int 23nums;`
(תו רווח בשם המשתנה)	`char ch ars;`
(סימן מינוס בשם המשתנה)	`float flt-num;`

בהגדרת המשתנים רצוי לציין מספר נקודות חשובות:

א. מומלץ להשתמש ב**שמות משמעותיים** כדי שהתוכנית תהיה קריאה ומובנת. יש לכך חשיבות בעיקר בתוכנית ארוכה ומורכבת, כאשר חוזרים לעיין בה לאחר זמן.

מתוך ההגדרות הבאות לא ניתן לדעת דבר על מהותם ותפקידם של המשתנים: i, c, j.

```
int i, j ;
char c   ;
```

ההגדרות הבאות מייחסות למשתנה **count** שימוש כמונה, למשתנה **sum** - שימוש כסכום של כמה מספרים, ולמשתנה **input_byte**, מטיפוס תו, כמקבל קלט ממקור כלשהו.

```
int count, sum   ;
char input_byte  ;
```

ב. שפת C **מבחינה בין אותיות קטנות לאותיות גדולות**, ולכן ההגדרה הבאה

```
int  COUNT, COunt, count ;
```

מתייחסת לשלושה משתנים **שונים**.

ג. ניתן לכתוב שם משתנה במספר תווים כלשהו, אך **המהדר מתייחס לכמות מוגבלת של תווים.** במרבית המהדרים יוכרו שישה עד שמונה התווים הראשונים. יש ללמוד את תכונותיו של המהדר, כדי למנוע אי הבנות כדוגמת המקרה הבא:

```
char student_num, students ;
```

במקרה זה, כאשר המהדר מתייחס רק לששת התווים הראשונים, שם המשתנה הראשון שיקלוט המהדר יהיה studen (שים לב, שישה תווים בלבד). על המשתנה השני תופיע הודעת שגיאה בזמן ההידור: "previously defined" שמשמעותה "הוגדר קודם", היות וגם הוא מתחיל בשש האותיות studen.

שיטות מתן שמות משתנים

למתן שמות משתנים חשיבות רבה להבנת הקוד אחרי זמן מה, או על ידי אנשים אחרים.

- **שמות קצרים למשתנים**: יעיל בכתיבה, אך קשה להבנה.
- **שמות ארוכים למשתנים**: יכול להועיל להבנה, אך מסרבל את הכתיבה.

מספר שיטות כתיבה נהוגות כיום:

א. השיטה ההונגרית

שיטה המקובלת לאחרונה, המתאימה לסביבת פיתוח הדורשת מספר רב של סוגי משתנים (Windows, OS/2). השיטה מבוססת על העיקרון של תחילית קצרה לשם

משתנה, כדי להצביע על הטיפוס שלו. אחרי התחילית, נרשם שם המשתנה באות גדולה. לדוגמה:

הסבר	שם המשתנה
מצביע לקובץ פלט	pfOutput
מערך תווים	szName
תו	chInput
אינדקס לולאה	iLoop

מקובל להשתמש בתחיליות הבאות:

Prefix	Data Type
a	Array (of any kind).
ch	Single Character.
cb	Count of bytes.
tp	Function Pointer or FILE pointer.
h	Handle (generic handle).
i	An index.
l	long integer.
lp	Far or large pointer.
n	Integer
np	Near or small pointer.
sz	ASCIIZ String (NUL Terminating String).
w	Word (unsigned int).

שיטה זו הונהגה בחברת Microsoft, ומאז היא משמשת כתקן מקובל בכתיבה למערכת ההפעלה Windows וגם בתכנות C++ בחברת Borland.

- ❖ בשיטה זו נהוג לתת שמות קבועים עם תחילית המעידה על טיפוס הקבוע ושאר האותיות גדולות.
- ❖ שמות פונקציות בנויות משילוב אותיות גדולות וקטנות על פי הכלל של אות ראשונה גדולה בכל מילה.
- ❖ במקרה של משתנים זמניים בפונקציות קצרות, ניתן להתעלם מהנוהל, או רצוי יותר להשתמש בשם משתנה המורכב מתחילית בלבד.

ב. השיטה השמרנית

אין זו שיטה אחת מגובשת, אלא אוסף סגנונות שהתפתחו בשינויים קלים במספר רב של מוסדות. השיטה גורסת:

- שמות משתנים באותיות קטנות בלבד עם הפרדה של מקף תחתון בין מילה למילה.
- שמות קבועים עם אותיות גדולות בלבד.
- שמות משתנים גלובליים בשילוב אותיות גדולות וקטנות.
- שמות פונקציות באותיות קטנות בלבד, מופרדות בקו תחתון בין מילה למילה.
- יש הגורסים כי גם לפונקציות יש לתת שמות המורכבים מאותיות גדולות וקטנות.
- במקרה של ריבוי משתנים גלובליים בפרויקטים גדולים, יש הנוהגים לכלול בשם המשתנה את שם המודול בו הוא מוגדר. יש אף ארגונים המוצאים לנכון לגבש נוהל משלהם לקביעת שמות לקובצי מקור.

במסגרת מערכת הפעלה DOS אין הדבר רלוונטי, מכיון ששמות קבצים מוגבלים למדי (8 תווים).

שיטה נוספת הנהוגה בחברת נובל: שמות משתנים מתחילים תמיד באותיות קטנות ואחר כך יש מעבר לשילוב אותיות גדולות קטנות.

קבועים

ראינו קודם שמשתנים מוגדרים על ידי שם וטיפוס. בתוכנית C אפשר להגדיר **קבועים** (Constants) מסוגים שונים:

קבוע מטיפוס מספרי

המהדר מייחס לכל קבוע **טיפוס** (type) הנקבע על ידי ערכו.

קבוע שלם (int)

נמצא בתחום 32,768- עד 32,768. מספרים שלמים החורגים מתחום זה ייחשבו כטיפוס long. מסיבה זו המספרים 102, 17-, 88 הינם קבועים מטיפוס int בעוד שהמספר 39,718- הינו קבוע מטיפוס long. ניתן להגדיר קבוע קטן יותר כטיפוס long על ידי הוספת התו "L" לדוגמה: 18L, 77L-.

קבוע שלם הקסדצימלי

ניתן לציין קבוע הקסדצימלי (בסיס 16) ע"י הקדמת הקבוע ב-0x (אפס והאות הקטנה x).

לדוגמה: 0x55, 0x37ac, 0xF73D.

קבוע שלם אוקטלי

כל קבוע שלם בעל הקידומת 0 (אפס) יחשב לקבוע אוקטלי (בסיס 8). לדוגמה: 037, 051, 072.

קבוע ממשי (real) ייחשב תמיד כ-double. קבועים כאלה ניתן לייצג במבנה עשרוני (דצימלי) רגיל, כגון: 12.37, ‎-15.1 או ייצוג **אקספוננציאלי** (E-Format) דהיינו: 1E-5, 32E-12.

מבנה אקספוננציאלי יוצג בצורה הבאה:

<כופל**>** **E** **<**חזקה עשרונית**>**

מכאן המספר 1E-5 פירושו $1*10^{-5}$ והמספר 32E-12 הוא $32*10^{-12}$.

קבוע מטיפוס תו

קבוע מטיפוס "תו" (char) מוגדר בקוד מוסכם בינלאומי מסוג ASCII, או מסוג EBCDIC. **קוד ASCII** (**A**merican **S**tandard for **C**oded **I**nformation **I**nterchange) הוא כינוי להגדרה בינלאומית של קודים לתווים. ההגדרה קובעת שכל תו מיוצג על ידי בית בודד (8 סיביות), על פי ערכו המספרי. לפיכך ניתן להגדיר בקוד זה עד 256 תווים שונים (קודים 0 עד 255). קוד ASCII מייחס לתווים הנומריים "0" עד "9" את הקודים המספריים 48-57 ולתו 'A' למשל, את הקוד המספרי 65. בנספח לספר תמצא את רשימת התווים וקוד ASCII שלהם. קוד ASCII מקובל במחשבים אישיים שונים, במחשבי דיגיטל ובמחשבים המותאמים לעבודה ביוניקס.

הגדרה אחרת של תווים נקראת קוד **EBCDIC** (**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode). הגדרה זו מקובלת במחשבי יבמ בארכיטקטורת 370/390 ומחשבים אחרים של יבמ.

בשפת C, **קבוע מטיפוס תו** נרשם כתו בודד, מוקף בגרשים בודדים. לדוגמה: 'A' מגדיר את התו A והמהדר מייחס לקבוע זה את ערך קוד ASCII (או EBCDIC בהתאם לסוג המחשב ומערכת ההפעלה) של התו. לדוגמה, ערכו של הקבוע 'B' במערכת הפועלת בקוד ASCII הוא 66, מכיון שזהו ערך ASCII של התו B. היתרון לשימוש בקבוע 'B' הוא בכך, שהוא יקבל ערך נכון גם אם נעביר את התוכנית למחשב הפועל בקוד אחר מקוד ASCII. דבר זה לא היה מתקיים, אילו כתבנו ישירות את הערך 66. דוגמה לשימוש אפשרי: עשרת התווים המייצגים את הספרות 0-9 הם בעלי קוד ASCII רציף. אם ברצוננו לקבל את הערך המספרי המתאים לכל תו כזה, נוכל לכתוב:

```
int num;
char c;
num = c - '0';
```

כאשר במשתנה c מטיפוס "תו" נמצא אחד מהתווים "0-9", נקבל ב-num ערך מספרי מתאים בתחום 0-9. שים לב כיצד ניתן להשתמש בקבוע "תו" בתרגיל חיסור.

בשפת C יש מספר קבועים מיוחדים מטיפוס "תו", אשר מופיעים בצמוד לתו "\" (Backslash - לוכסן הפוך), כדי להבדילם מתווי ASCII רגילים. כאשר המהדר נתקל בצמד התווים \<תו> (כלומר, לוכסן הפוך ומשמאלו תו מיוחד) - הוא מפענח אותם כתווי בקרה. התווים המיוחדים הינם: **n, r, b, t, f, ' ו-**\. על משמעותם נעמוד להלן.

א. '\n' - שורה חדשה - Newline (קוד ASCII 10):

תו זה יוחלף על ידי המהדר בתו הבקרה "Newline" שקוד ASCII שלו הוא 10. בפקודת הדפסה יגרום תו זה למעבר הסמן (Cursor) לתחילת שורה חדשה.

ב. '\r' - חזור לתחילת שורה - Carriage Return (קוד ASCII 13):

בפקודת הדפסה, תו זה ימקם את הסמן בתחילת השורה בה הוא נמצא.

ג. '\t' - טבלר (טבולטור) - Tab (קוד ASCII 9):

בפקודת הדפסה תו זה יגרום ליצירת מרווח, בגבול אינטגרלי של 8 מהמקום בו נמצא הסמן. המסך, או דף ההדפסה, מחולקים באופן לוגי לעמודות בנות 8 מקומות כל אחת, החל מצד שמאל. הטור הראשון בתחילת כל עמודה הינו: 1, 9, 17, 25, ... ניתן לשנות את גודל המרווח באופציות המהדר.

לדוגמה, אם הסמן נמצא במקום 12 במסך, תגרום פקודת ההדפסה עם התו '\t' לתזוזת הסמן למקום 17.

ד. '\b' חזור תו אחד - Backspace (קוד ASCII 8):

תו זה יגרום לנסיגה של הסמן תו אחד לאחור.

ה. '\f' דף חדש - Form Feed (קוד ASCII 12, במדפסת בלבד):

בפקודת הדפסה, תו זה יגרום לקידום דף שלם במדפסת, או לניקוי המסך.

ו. לתווים "גרש" (') ו"לוכסן הפוך" (\) יש משמעות מיוחדת ולכן, כדי להגדיר אותם כקבוע, יש להשתמש ב-\ נוסף.

למשל: קבוע התו \ יוגדר על ידי \\.

קבוע התו ' יוגדר על ידי '\.

ז. '\nnn' - כדי להגדיר תו, באמצעות ערכו המספרי, משתמשים בהגדרה הכוללת את תו Backslash והערך האוקטלי של התו:

nnn - הן הספרות האוקטליות המייצגות את ערך ASCII של התו. לדוגמה, התו 'A' שערכו המספרי 65 וערכו האוקטלי 101, מיוצג על ידי הצירוף "\101".

ח. \xnn - הגדרה של תו באמצעות ערכו ההקסדצימלי nn. לדוגמה, \x30 מגדיר תו שערכו ההקסדצימלי 30.

ט. התו שערך ASCII שלו 0 (אפס) הינו תו NULL המשמש בין השאר, כתו המסיים מחרוזות.

קבוע מטיפוס מחרוזת

מחרוזת היא אוסף של תווים אלפא-נומריים כלשהם המהווה יחידה אחת. קבוע מחרוזת יוגדר על ידי ציון של גרשים (") משני צדדיו. לדוגמה:

```
"I am learning c!"
```

בניגוד לקבועים הקודמים, מוגדר עבור תווי המחרוזת מקום בזיכרון, וערך הקבוע הוא הכתובת של אותו מקום. באותה כתובת נמצא את ערכי ASCII של התווים המרכיבים את המחרוזת ובסופם - את תו NULL שהוזכר קודם. את תו NULL מוסיף המהדר עבורנו בסיום כל קבוע מחרוזת.

ניתן לשלב בתוך המחרוזות תווים מיוחדים (המופיעים בצמוד ל-\), או ערך אוקטלי של קוד ASCII (של תו רצוי), גם-כן בצמוד ל-\.

לדוגמה:

המחרוזת "This string \n Occupies 2 lines", מוכלת ב-2 שורות הדפסה נפרדות, כפי שהיא מעידה על עצמה, כי משובץ בה התו המיוחד n\ - שפירושו: שורה חדשה. המחרוזת הזו:

```
"The digits are: \036\037\040\041\042\043\044\045\046\047"
```

מייצגת למעשה את התווים האלה:

```
"The digits are: 0123456789"
```

תווי הספרות 0-9 מיוצגים במחרוזת המקורית על ידי קוד ASCII שלהם, בבסיס אוקטלי. דוגמה: אם המחרוזת "abc" נמצאת בכתובת 6000, תמונת הזיכרון תהיה:

התו	ערך ASCII	כתובת
'a'	97	6000
'b'	98	6001
'c'	99	6002
'\0'	0	6003

טיפוס enum

טיפוס enum מיועד להגדרת קבוצת קבועים מסוג int. ערך ברירת המחדל של הקבוע הראשון הוא 0. המבנה:

enum {<constant_name> [=<value>], ...};

constsnt_name שם הקבוע.

value ערך מסוג integer.

דוגמאות:

```
enum {SUNDAY=1, MONDAY, TUESDAY};
enum {a=3, b, c, d=8, e};
```

כך נציג את ערכי enum:

```
printf("%d, %d, %d /n", SUNDAY, MONDAY, TUESDAY);
```

הפלט:

```
1, 2, 3
```

```
printf("%d, %d, %d, %d, %d /n", a, b, c, d, e);
```

הפלט:
```
3, 4, 5, 8, 9
```

שים לב!

כאשר לקבוע מסוים חסר ערך, ערכו הקודם יהיה: הערך הקודם 1+.

אופרטורים

עד כה הכרנו את הטיפוסים הקיימים בשפת C, משתנים וקבועים, תחומים ושיטות ייצוג. אלה הם אבני הבניין של השפה. עתה עלינו ללמוד כיצד להשתמש בהם לשם ביצוע פעולות עיבוד שונות.

במשפט בשפת C מבחינים בשתי קבוצות שונות: **קבוצת אופרנדים** (או נפעלים), שכוללת את המשתנים והקבועים, ו**קבוצת אופרטורים** (או מפעילים), שכוללת את סימני הפעולה בין האופרנדים.

בסעיף זה נדון במפעילים שבשפה ובקבוצות המשניות המחלקות אותם: אופרטורים אריתמטיים, אופרטורים לוגיים ואופרטורים לוגיים בסיביות.

אופרטורים אריתמטיים

שפת C מאפשרת חמש פעולות אריתמטיות בסיסיות:

חיבור	+
חיסור	-
כפל	*
חילוק	/
שארית (מודולו)	%

האופרטורים האלה הם **אופרטורים בינאריים**, דהיינו, מימין ומשמאל לסימן הפעולה ישנם **אופרנדים**.

לדוגמה: `a + 5 ,X/Y ,num % 3`

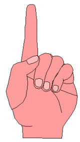
ניתן להשתמש באופרטור החיסור כ**אופרטור אונרי**, המופיע משמאל לאופרנד, ואז יהיה משמעות שלילת ערך האופרנד.

לדוגמה: `-a * 6`

הירכיה של פעולות אריתמטיות

הביטוי האריתמטי הוא ביטוי הכולל בתוכו אופרטורים אריתמטיים ואופרנדים. סדר הביצוע של הפעולות תלוי במוסכמות של קדימות, אשר קבועות בשפה. הטבלה הבאה מתארת את סדרי הקדימויות בהערכת ערכו של ביטוי בשפת C.

קדימות	אופרטור
1	()
2	- יחידני
3	%, /, *
4	-, +
5	=



שים לב!

לסוגריים יש רמת קדימות גבוהה ביותר ולכן, ניתן באמצעות סוגריים לשנות את סדר הקדימות של שאר האופרטורים.

נציג מספר דוגמאות כדי להמחיש את המשמעות של חוקי הקדימות:

א. **a + b / d - 10**

בביטוי זה, סימן החילוק קודם לסימני החיבור והחיסור. לכן תחושב תחילה המנה b/d ורק אחר כך תבוצענה פעולות החיבור והחיסור. מכיון שביטוי אריתמטי מוערך משמאל לימין, תבוצע פעולת החיבור לפני פעולת החיסור.

ב. **(a + b) / d - 10**

בביטוי זה הסוגריים הם בעלי העדיפות הגבוהה ביותר, ולכן יחושב תחילה הסכום a+b. פעולת החילוק היא השנייה בסדר העדיפות ולכן הסכום a+ b יחולק בערך המשתנה d, ולבסוף תבוצע פעולת החיסור.

אנו ממליצים להשתמש בסוגריים בכל מקום שבו הערכת הביטוי אינה חד-משמעית, או כאשר הביטוי מורכב ומכיל פעולות רבות.

ג. **b + d * num / e - upper + 10 / lower**

נכתוב ביטוי זה שנית, והפעם - נשתמש בסוגריים וניווכח שהוא מובן יותר:

b + ((d * num) / e) - upper + (10 / lower)

בביטוי זה יחושבו תחילה הביטויים שבסוגריים, ולכן המכפלה d*num תחושב תחילה, אחר כך תבוצע החלוקה ב-e ותחושב המנה 10/ lower, ולבסוף תבוצענה פעולות החיבור והחיסור.

אופרטורים לוגיים

סוג אחר של ביטויים הקיימים בשפה מכיל **אופרטורים לוגיים** (Logical Operators). ביטויים אלה כוללים פעולות המבוצעות בין משתנים או קבועים, אשר תוצאתן היא הערך "אמת" או "שקר" (true או false בהתאמה).

ביטוי לוגי מכיל מספרים או משתנים מקושרים ביניהם באמצעות המפעילים הלוגיים. על עיקרון זה מבוססים משפטי התנאי הקיימים בשפה. ניתן, כפי שנראה בהמשך, לקשר בין מספר ביטויים לוגיים (בדומה לפעולות אריתמטיות) כדי ליצור ביטוי לוגי מורכב. גם תוצאתו של ביטוי כזה היא "אמת" או "שקר". נזכיר כאן, שבשפת C כל מספר **שונה מ-0** הוא בעל ערך לוגי "**אמת**", ורק המספר **0** הוא "**שקר**".

האופרטורים הלוגיים ומשמעותם מוצגים בטבלה הבאה:

אופרטור	הסבר
<	קטן מ-
>	גדול מ-
==	שווה ל-
<=	קטן או שווה ל-
>=	גדול או שווה ל-
!=	לא שווה ל- (שונה מ-)

דוגמאות:

א. "a גדול או שווה ל-b":

```
a >= b
```

תוצאת ביטוי זה תהיה "אמת" (ערך מספרי שונה מ-0), אם a גדול או שווה ל-b.

ב. "x שווה ל-y":

```
x == y
```

תוצאת ביטוי זה תהיה "אמת" אם x שווה ל-y, ו"שקר" (ערך מספרי 0) - אם x אינו שווה ל-y.

כפי שהזכרנו קודם, ניתן לקשר בין כמה ביטויים לוגיים כדי ליצור ביטוי לוגי מורכב. פעולת הקישור נעשית באמצעות אופרטורים המפורטים בטבלה הבאה:

<table>
<tr><th>הסבר</th><th>אופרטור</th></tr>
<tr><td>היפוך (not)</td><td>!</td></tr>
<tr><td>וגם (and)</td><td>&&</td></tr>
<tr><td>ו/או (or)</td><td>||</td></tr>
</table>

אופרטור ההיפוך קודם בעדיפותו על האופרטורים האחרים המשמשים לקישור (למעשה גם האופרטורים הלוגיים קודמים לאופרטורי הקישור).

א. הביטוי - "**לא** (x **שווה ל-**y)":

```
!(x == y)
```

הביטוי יהיה אמת, אם x שונה בערכו מ-y. ביטוי זה מקביל לביטוי הלוגי x != y.

ב. "x שווה ל-y וגם a גדול או שווה ל-b":

```
x == y && a >= b
```

כדי שהביטוי יהיה ברור יותר, מומלץ להשתמש בסוגריים:

```
(x == y) && (a >= b)
```

ביטוי זה יהיה אמת, רק אם ערכו של x שווה לערכו של y וגם ערכו של a גדול או שווה לערכו של b.

ג. כדי להקדים בביטוי פעולה לוגית אחת לאחרת, נשתמש בסוגריים בדומה לדרך שבה נוהגים בפעולות אריתמטיות. לדוגמה,

```
((c >= s) || (b == a)) && (c != 0)
```

במקרה זה, יוערך תחילה הביטוי שבסוגריים:

"c גדול או שווה ל-s או b שווה ל-a", ואחר כך יוערך הביטוי "וגם c אינו שווה אפס". תוצאת הביטוי שלהלן תהיה זהה לתוצאה של הביטוי הקודם:

```
((c >= s) || (b == a)) && c
```

הזהות קיימת, מכיון שבשפת C כל ביטוי **שערכו שונה מאפס** הוא **אמת**!

הערכת ביטוי לוגי

הערכה של ביטוי לוגי נעשית משמאל לימין, בדומה להערכה של ביטוי אריתמטי. הערכת הביטוי נעשית בשיטה של ***"מעגל קצר"*** (Short Circuit) כלומר, ההערכה ממשיכה עד שערכו של הביטוי נקבע בוודאות.

בביטוי שמכיל רק פעולות "**או**" (||) - די בגורם בודד שיהיה אמת, כדי שהביטוי כולו יהיה אמת. בביטוי שמכיל רק פעולות "**וגם**" (&&) - די בגורם "שקר" אחד כדי שהביטוי כולו יהיה שקר.

לדוגמה, נתון הביטוי

```
(a <= b) || (c == d)
```

בביטוי זה אין צורך לבדוק את השוויון של c ל-d, אם מצאנו כי a קטן או שווה ל-b, מכיון שערכו של הביטוי כבר נקבע כ-true. בביטוי הבא,

```
(x < y) && (f != g) && (w >= 3)
```

אם x גדול או שווה ל-y, הערכת הביטוי תיפסק ושאר ההשוואות לא תבוצענה, כי ברור שהערך של כל הביטוי יהיה שקר. לעובדה זו חשיבות רבה במיוחד כפי שנראה בהמשך, כאשר נשלב בביטויים לוגיים גם פעולות אריתמטיות.

לדוגמה:

```
(a <= b) && ((x = y + b) > 3)
```

במקרה כזה, אם a גדול מ-b **לא תבוצע פעולת החישוב וההשמה למשתנה x**, מכיון שאין במקרה זה צורך להעריך את שארית הביטוי. מהלך התוכנית עלול להשתבש אם בהמשך התוכנית הסתמכנו על ערכו של x.

אפשר לפתור בעיה זו, אם נהפוך את סדר הארגומנטים:

```
((x = y + b) > 3) && (a <= b)
```

אולם, לא תמיד קיים פתרון כזה. לדוגמה:

```
((x = y + 3) < 10) && ((a = b) == 5)
```

עד עתה למדנו ששפת C יכולה לטפל בסוגים שונים של נתונים: **תווים** (characters) **ומספרים** מסוגים שונים, כמו, מספרים שלמים, עשרוניים רגילים וארוכים. עתה נתבונן בסוגי הייצוג השונים של נתונים מנקודת מבט של החומרה.

מילים שמורות

להשלמת הדיון במרכיבים של שפת C מובאת להלן רשימה של **"המילים השמורות"** (Reserved Words) של השפה. השימוש במילים אלו מוגבל להגדרות ולבנייה של תוכניות, ולא ניתן להשתמש בהן בשמות של משתנים וקבועים:

```
auto       default    extern    int       sizeof    union
break      do float   long      static    unsigned
case       double     for       register  struct    while
char       else       goto      return    switch    const
continue   entry      if        short     typedef   enum
signed     void       volatile
```

הערות

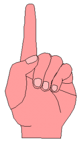
הערות (Comments) נועדו להסביר ולציין קטעים או משפטים בתוכנית. פעמים רבות, חשיבות הערות אינה נופלת מהמשפטים של התוכנית והן עוזרות להבין ולבדוק אותה.

כאשר יש לכתוב הערות בגוף התוכנית, תוחמים אותן בסימנים "*/" ו-"/*" מימין ומשמאל בהתאמה. הערה בשפה יכולה להיכתב בכמה שורות ולהכיל כל תו אפשרי. רק כאשר ייתקל המהדר בצירוף הסימנים "/*" הוא יסיים את ההערה וימשיך בתהליך ההידור. המהדר אינו מתייחס כלל לתווים שבגוף ההערה.

דוגמאות:

```
int  num,count  ;  /* counters for do-while loops */
char chr;          /* variable to store input
                      from  keyboard */
        /* this*/
        /* is an example */
```

בזמן בדיקת תוכניות ניתן לכלול קטעים שלמים בהערה, כדי להקטין זמנית את נפח התוכנית ולטפל רק בקטעים שבהם מתכוונים לאתר את מקור השגיאה. דוגמה לכך תינתן בהמשך.



שים לב!

במהדרים של בורלנד החל מגירסה 3.1 ניתן לסמן שורת הערה בסימון //. כל התווים החל מסימון זה ועד סוף השורה ייחשבו כהערה. הערה זו אינה יכולה לגלוש מהשורה שבה היא התחילה אל שורה נוספת

תרגילים

תשובות בתקליטור: קבצים C:\tclite\targilim\c1e1-c1e12

1. סווג את המספרים והקבועים הבאים לפי הטיפוסים בשפת C:

```
1987     352741     0x7F
10.7     85         5555
-15      77.56213   32500
c        1252751     33000
-75000   18L        1E-5
025
```

2. לפניך שמות של משתנים. סמן את השמות החוקיים.

```
count    -1num      begin      dobegin
c ount   &show      i          7 num
1num     show&      !count     result
num1     ?if        while1
100find  if?        co unt
```

3. לפניך רשימה של משתנים. כתוב הגדרות מתאימות עבורם:

א. מונה לולאה.

ב. מונה פעימות מונה לטלפון.

ג. יתרה בחשבון בנק.

ד. מספר שלם בתחום 0 :52000-.

ה. מספר שלם בתחום 32000 :32000-.

ו. אות מיון במדריך טלפונים.

ז. מספר טלפון.

ח. ספרה במספר.

ט. משתנה בוליאני (אמת או שקר).

4. מהם קבועי התו המיוחדים לשפת C?

5. כיצד תגדיר את קבועי התו גרש (') וקו נטוי (/)?

6. האם קיים הבדל בין הקבוע 57 והקבוע "9" (בהתחשב בעובדה שערך ASCII של הספרה 9 הוא 57). הסבר.

7. השלם את הטבלה הבאה:

ערך הקסדצימלי	ערך אוקטלי	ערך דצימלי
3F		
	20	
		15
	103	
77		
	205	
		2001

8. כיצד תייצג את הביטויים הבאים בעזרת האופרטורים שבשפה:

$$\frac{52 * (7a + 3b)}{5c} \qquad \frac{8t + 7b}{\frac{3a}{5} + 2c} \qquad \frac{X +3 + (Y * Z)}{X *Y * Z} \qquad \frac{X Z}{5} - t$$

9. הפוך את המשפטים הבאים לביטויים המתמטיים המתאימים:

```
8*t + 7*b/3*a/5+2*c      x/y/z+5*a        -X+3*Y-5/a
                         x/(y/z) + (5*a)  (3-x*7/y)*a+6
```

10. כתוב את הביטויים הלוגיים המתאימים למשפטים הבאים:

א. a גדול מ-b.

ב. b קטן או שווה ל-a (מצא שני ביטויים שיתאימו למשפט זה).

ג. c שווה ל-b, וגם b קטן מ-a.

ד. x שונה מ-y, או y קטן או שווה ל-z.

ה. x קטן מ- (a*5).

ו. r שונה מ-p וגם [z קטן או שווה ל-y או y גדול מ-x].

11. מהו הערך הלוגי של הביטויים הבאים:

א. `(x > y) && (y > x)`

ב. `(x == y) || (y != x)`

ג.

```
(y > x) && ((w == y) || (x < 3))
```

כאשר:

```
w = 25 ,z = 9 ,y = 5 ,x = 7
```

ד.

```
r == ((p < m) && ((n > r) || (p < n)))
```

כאשר:

```
r = 0 ,p = 13 ,n = 12 ,m = 109
```

12. חזור לתרגיל 11 (סעיפים ג׳, ד׳) וציין, באיזה מקום תיפסק הערכת הביטוי בזמן הריצה. מצב זה חל כאשר ערכו של הביטוי נקבע באופן מוחלט לפני שמגיעים לסופו.

פרק 2

משפט ההצבה ומבנה התוכנית

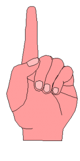
תוכנית מחשב בשפת C, בדומה לתוכנית בשפות תכנות אחרות, מורכבת מאוסף של **משפטים** (Statements) של השפה. בכל שפה נקבעת חוקיות המשפטים על פי חוקי דקדוק וחוקי תחביר. כך גם בשפות מחשב ובכללן - **בשפת C**.

בסעיפים הבאים נעסוק בכללי ה**תחביר** (Syntax) של משפטים בשפת C, ובמקביל יוסברו תכונותיהם ואופן השימוש בהם.

מבנה המשפטים ומבנה כללי של תוכנית

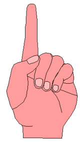
משפט בשפת C מגדיר פעילות חישובית או לוגית שיש לבצע במחשב. בדרך כלל הוא מכיל **מילת מפתח** שמורה (Keyword), המגדירה את סוג המשפט, ביטוי אחד או יותר וסימני פיסוק מיוחדים. כל משפט חייב להסתיים ב-"**;**" (נקודה-פסיק). משפט אחד יכול להיכתב במספר שורות ללא סימון מיוחד בסיומה של כל שורה, בתנאי שבסופו יופיע הסימן "**;**". המשפטים מופיעים ברצף, זה אחר זה, ללא מספור שורות, והסדר שלהם נקבע על פי מיקומם בתוכנית. בדוגמאות בספר זה צוינו מספרי שורות **רק** כדי שאפשר יהיה להתייחס למשפטי התוכנית בעת ההסבר.

לשם כתיבת תוכנית משתמשים ב**עורך תמליל** (Text Editor) או ב**מעבד תמליל** (Word Processor, שמפיק גם טקסט ASCII ללא סימני עריכה של המעבד). בעזרתם מכינים קובץ תוכנית המכיל את המשפטים המרכיבים אותה. בדרך כלל משתמשים בעורך התמליל של ערכת הפיתוח.



שים לב!

אין להשתמש במעבד תמלילים, מכיון שהוא מפיק גם קודים מיותרים, שאינם דרושים כאן.



שים לב!

***אין מגבלה** למספר הרווחים המופיעים בין המרכיבים השונים של המשפט: ביטויים, משתנים, מילות מפתח וסימני פיסוק. רצף רווחים כזה יזוהה על ידי המהדר כרווח יחיד, אך ניתן לנצל עובדה זו בשעת כתיבת התוכנית לסידור השורות, כדי שיקל עלינו לקרוא אותה אחר כך.*

מספר משפטים פשוטים יכולים להיות מקובצים כ**יחידה לוגית אחת - בלוק**, הנקראת **משפט מורכב** (Complex Statement).

תוכנית מחשב מורכבת בדרך כלל ממספר יחידות לוגיות, או תת-תוכניות. תת-תוכניות אלו יכולות, במקרים מסוימים, להכיל משפט או משפט מורכב בודד, ובמקרים אחרים - עשרות משפטים כאלה. בשפות תכנות שונות אפשר להגדיר תת-תוכניות מסוגים שונים, בעלות תכונות ואפיונים שונים: **פרוצדורה** (Procedure), **פונקציה** (Function), **שיגרה** (Subroutine או Routine) ועוד. בהמשך נשתמש במושג **פונקציה** בלבד, כדי לתאר ולציין תת-תוכניות בשפת C.

לכל פונקציה קיים שם מזהה, בדומה לשם משתנה, והוא חייב להיות יחידני באותה תוכנית. **שם פונקציה** רושמים בדרך זו:

<שם הפונקציה> **()**

למשל: ()xyz. הסוגריים מסמנים למהדר שהשם הוא שם של פונקציה ולא שם משתנה.

כל תוכנית חייבת להכיל פונקציה אחת בעלת השם השמור: **()main**. פונקציה זו מופעלת אוטומטית כאשר מריצים את התוכנית. שיטה זו מאפשרת להכיל את כל התוכנית בפונקציה ()main, או לקרוא מתוך ()main לפונקציות נוספות.

המשפטים המרכיבים פונקציה מתוחמים בזוג סוגריים מסולסלים ("{" ו-"}") המקבילים לצירוף begin-end בשפת פסקל או PL/M. לפניך תיאור סכימטי של תוכנית C בסיסית:

```
void main (void)
{
   משפט ;
   משפט ;
    ...
   func_1();
    ...
   משפט ;
}

void func_1(void)
{
   משפט ;
    ...
   משפט ;
}
```

()func_1 היא פונקציה, אשר מופעלת מתוך הפונקציה ()main. שים לב לכך שהגדרתה נעשית בנפרד, **מחוץ** לפונקציה הקוראת. כמו כן, שים לב לסוגריים "()" הצמודים לכל שם פונקציה (במקרה זה - ()main ו-()func_1).

משפט הצבה

תפקיד **משפט ההצבה** לקבוע ערך למשתנה. התבנית הכללית של משפט הצבה:

<משתנה> = <ביטוי> ;

המשתנה חייב להיות מוגדר לפני משפט ההצבה. סימן השוויון "=" מפורש כאופרטור הצבה, ולא כשוויון אריתמטי. הפקודה מבוצעת מימין לשמאל: תחילה מחושב הערך של הביטוי, התוצאה מתורגמת לטיפוס של המשתנה שמשמאל לסימן השוויון, ואחר כך היא מוצבת לתוכו. דוגמה:

```
int  num ;
char ch ;
...
num = ch + 3 ;
```

תחילה יחושב הביטוי ch + 3 ויישמר בתוך אזור זמני בזיכרון והתוצאה שהיא מטיפוס "**תו**" (char), תּמַר לטיפוס "**שלם**" (int), אשר יוצב לתוך המשתנה num. לדוגמה, הפקודה:

```
ch = num ;
```

תמיר את ערך המשתנה num לטיפוס תו ותציב ערך זה במשתנה ch. ההמרה ממספר שלם לתו (שגודלו בזיכרון קטן מגודל הטיפוס "שלם") יתבצע כך, שערכו של הביטוי לא ישתנה, בתנאי שאינו עולה בערכו על הערך המקסימלי של תו (255).

ההמרה של טיפוס הביטוי נעשה לפי הטיפוס של המשתנה המקבל את תוצאת ההצבה. נציג את החוקים לכך בטבלה 2.1.

משפט ההצבה עצמו נחשב מבחינה תחבירית, כביטוי שערכו הוא הערך שהוצב למשתנה שמשמאל לסימן השוויון. מסיבה זו ניתן לכתוב **משפט הצבה מורחב,** בו הביטוי **מימין** לסימן "=" הינו משפט הצבה. בדרך זו ניתן לבצע "הצבה מרובה" למספר משתנים. דוגמה:

```
int  x ;
int  i, j ;
...
x = 7 ;
i = j = 5 * x ;
```

המשתנה j קיבל את ערך הביטוי 5*x שערכו 35. המשתנה i יקבל את תוצאת הביטוי (j = 5 * x), שגם ערכו הוא 35.

טבלה 2.1: כללי הסבה לפעולות עם משתנים מעורבים

אופן התרגום	טיפוס הביטוי המחושב	טיפוס המשתנה המקבל
ערך הביטוי, מודולו 256, התייחסות ל-byte	char	int
ערך הביטוי, מודולו 256, תוך קיצוץ של השבר העשרוני	float\double	char
ערך זהה	char	int
ערך הביטוי, מודולו 32,768 תוך קיצוץ של השבר העשרוני	float\double	int
ערך זהה, שבר עשרוני = 0	char\int	float
חצי של הספרות המשמעותיות (קיצוץ מנטיסה)	double	float
ערך זהה, שבר עשרוני = 0	char\int	double
ערך זהה	float	double

תרגיל לחזרה

לסיום הדיון בסעיף זה נציג תוכנית פשוטה המיישמת את הנושאים שלמדנו. התוכנית מחשבת משכורת נטו של העובד בהתחשב במס שעליו לשלם. התוכנית בנויה משתי פונקציות: פונקציה ()main (ההכרחית) ופונקציה ()calc_tax. אין צורך בשלב זה להבין במדויק את מהלך התוכנית, אלא יש להתרשם מהמבנה שלה: הדרך בה רשומים משפטי ההצבה, החלוקה לפונקציות והדרך שבה נכתבים המשפטים.

נתון תלוש משכורת המכיל את הנתונים הבאים:

hours	משתנה מטיפוס שלם המכיל את מספר שעות העבודה החודשיות.
hour_pay	משתנה מטיפוס ממשי המכיל את השכר לשעת עבודה.
tax	משתנה מטיפוס תו המכיל את אחוז המס על התשלום עבור שעות עבודה. בחרנו בטיפוס "תו", מכיון שטווח התו (127+, 127-) מספק לצורכי הייצוג של האחוזים (0-100).
years	משתנה מטיפוס תו המכיל ותק העובד.
year_pay	התשלום עבור שנת ותק, משתנה זה הוא מטיפוס ממשי. על התשלום עבור שנות ותק אין משלמים מס.
salary	משתנה מטיפוס שלם המכיל את הערך השלם של המשכורת.
calc_tax()	זו פונקציה המחשבת את ערך המס על המשכורת.
tax_pay	משתנה מטיפוס שלם המכיל את ערך המס המחושב.



הערה!

*המספרים שבסוגריים, אשר נכתבו משמאל למשפטי השפה, נרשמו **לצורך הסבר בלבד, ואין הם דרושים** בכתיבה סדירה של תוכניות בשפת C.*

להלן התוכנית:

```
/*****      chp0201.c  :קובץ תקליטור       *****/

#include <stdio.h>

(1)     void calc_tax(void);
(2)     int hours, salary, tax_pay;
(3)     float hour_pay, year_pay;
(4)     char years, tax;

(5)     void main(void)
(6)     {
(7)       hours = 275;
(8)       hour_pay = 32.5;
(9)       tax = 35;
(10)      calc_tax();
(11)      years = 5;
(12)      year_pay = 10.25;
(13)      salary = (hours*hour_pay)+(years*year_pay)-
          tax_pay;
(14)      printf ("המשכורת נטו : %d\n",salary);
(15)    }

(16)    void calc_tax (void)
(17)    {
(18)      tax_pay = hours*hour_pay*tax/100;
(19)    }
```

בתוכנית זו מוצגים מספר נושאים חדשים, כגון פונקציות וטווחי הכרה, אותם נסביר בהמשך. למרות זאת, ניתן להבין את מהלכה ואת הפעולות העיקריות שבה.

- במשפט 1 מתבצעת הכרזה של הפונקציה calc_tax.
- במשפטים 2-4 מוגדרים משתני התוכנית. נתנו להם שמות משמעותיים כדי להקל על הבנת התוכנית והקצנו לכל אחד מהם את הטיפוס המתאים לו ביותר. זהו כלל ראשון וחשוב בתכנות נכון ומסודר.
- בשורות 5-15 אנו מגדירים את הפונקציה הראשית - main().

בתחילתה (שורות 9-7) אנו מציבים את הערכים למשתנים המשתתפים בשלב החישוב הראשון. בדרך כלל שלב ההצבה נעשה לאחר קליטת הערכים בפעולות קלט/פלט, אך אותן טרם למדנו.

בשורה 10 מבוצעת קריאה לפונקציה ()calc_tax כדי שתחשב את המס שעל העובד לשלם.

כעת נותר לנו לחשב את התוספת עבור הוותק ולהציב את התוצאה הסופית במשתנה "משכורת" (salary) (שורות 13-11). שים לב, שתוצאת הביטוי הימני בשורה 13 היא מטיפוס ממשי (float) ובפעולת ההצבה מבוצעת המרה לטיפוס שלם, לפי החוקים שלמדנו.

הסבה בכוח

פעולת **הסבה בכוח** היא המרת טיפוס של משתנה באופן זמני בזיכרון, כדי לערוך חישוב מסוים כשאיננו רוצים להוסיף משתנה נוסף, או כדי להמיר בין טיפוסים ולהימנע באופן מודע מאיבוד ערכו האמיתי של המשתנה.

לדוגמה:

הפונקציה Sqrt(x) המבצעת פעולת חישוב שורש ריבועי. לפי הגדרת הפונקציה, x חייב להיות משתנה מטיפוס double. בהנחה ש-x מוגדר כ-int לצורך פעולות חישוביות אחרות ואיננו רוצים להגדיר משתנה נוסף מטיפוס double, אנו מבצעים את הפעולה הבאה:

```
int x,result;
x=a;
result=sqrt((double)x);
```

בפעולה זו אנו מאלצים את x להפוך לטיפוס double באופן זמני בזיכרון. פעולה זו נקראת **Casting**. התבנית:

(<שם הטיפוס>) <ביטוי>

דוגמאות:

```
(int) chr
(long)(x+7)
(float)((int)chr * i)
```

פעולת casting היא למעשה אופרטור יחידני ותכונותיה, ובכללן חוקי הקדימות, דומות לאופרטורים כגון"*" ו-"&". בתוכניות שנכתוב בהמשך נשתמש ב-casting במקרים הבאים:

א. כאשר משתמשים בביטוי, או ערך המוחזר מפונקציה בהצבה למשתנה מטיפוס אחר.

ב. כאשר קוראים לפונקציות עם ערכים מטיפוסים שונים מההגדרה.

ג. כאשר מעוניינים להבטיח שהמהדר אכן יבצע המרה מסוימת (גם אם עקרונית עליו לעשות זאת). פעולה זו נועדה לתמוך בתאימות התוכנית לגבי מהדרים שונים, אשר חלק מהם אינם כוללים אפשרות זו.

הגדלה והקטנה עצמית של משתנים

בשפת C יש דרך נוספת להציב ערכים למשתנים. בדרך זו אין שימוש מפורש במשפט ההצבה. שיטה זו נקראת **הגדלה והקטנה עצמית** (Auto increment/decrement), והיא מהווה למעשה שיטת כתיבה מקוצרת של ביטויים נפוצים. דוגמה:

```
int  i;
i = i + 1 ;
i = i - 1 ;
```

ביטויים אלה מיועדים להגדיל או להקטין ערך של משתנה ביחידה אחת.

הכתיבה המקוצרת תהיה, בהתאמה:

```
i++ ;
i-- ;
```

ניתן להשתמש בכתיבה המקוצרת למשתנה מכל טיפוס כלשהו.

השילוב של ביטויים אלה בביטויים מורכבים מאפשר גמישות ויעילות בתכנות. דוגמה פשוטה לכך נראה בפעולות ההצבה הבאות:

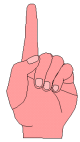
```
(1)  int i, j ;
(2)  i = 5 ;
(3)  j = ++i + 3 ;
```

בשורה (3) שילבנו הגדלה עצמית בפקודת הצבה. לאחר ביצועה ערכו של i יהיה 6 ושל j יהיה 9.

ההגדלה וההקטנה העצמית מופיעות בשתי צורות:

קדם הגדלה/הקטנה (Pre increment/decrement) `++i ,--i`

הגדלה/הקטנה מאוחרת (Post increment/decrement) `i++, i--`



שים לב!

*בשתי דרכים אלו הערך הסופי של i יהיה **זהה** אבל ערך הביטוי **לא** יהיה שווה בשני השימושים השונים. הנה ההסדר: **בהגדלה מאוחרת** יחושב הביטוי המורכב באמצעות הערך הקודם של המשתנה ואחר כך תתבצע ההגדלה. תהליך **קדם הגדלה** מבצע את ההגדלה תחילה, ורק לאחר מכן יחושב הביטוי המורכב על פי ערכו החדש של המשתנה. כך גם לגבי הקטנה עצמית.*

הדוגמה הבאה ממחישה את השימוש בדרך כתיבה זו :

```
(1)  int i = 5,r ;
(2)  r = 3 + (7 * i++) ;
```

בשורת ההצבה (2), המספר 7 הוכפל ב-5 (ערך i לפני ההגדלה) ולכן, ערכו של r יהיה 38 וערכו של i יהיה 6.

```
(3)  i = 5 ;
(4)  r = 3 + (7 * ++i)
```

בשורת ההצבה (4), המספר 7 הוכפל ב-6 (ערך i אחרי ההגדלה) ולכן, ערכו של r יהיה 45 וערכו של i יהיה גם כאן 6.

משפט הצבה מקוצר

במקרים רבים נמצא בתוכנית משפט, המכיל את **אותו המשתנה** משני עברי סימן ההצבה. למשל, נתון משפט בעל המבנה הבא :

```
<משתנה_1> = <משתנה_1> <פעולה> <ביטוי>  ;
```

שפת C מאפשרת תחביר מקוצר למשפט בעל מבנה כזה :

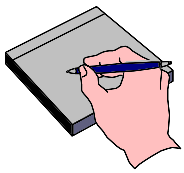
```
<משתנה_1> <פעולה>= <ביטוי>  ;
```

המשפטים :

```
x = x + y ;
i = i * (8 + j) ;
chr_1 = chr_1 & (0x01 || chr_2) ;
```

שקולים למשפטים :

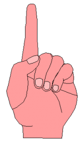
```
x += y ;
i *= 8 + j ;
chr_1 &= 0x01 || chr_2 ;
```



הערה!

*כאשר קוראים תוכניות בשפת C צריך לשים לב לכתיבה מקוצרת. למשל, למשפט ההצבה x += y יש משמעות **שונה** בתכלית מזו שיש למשפט ההצבה x = y.*

המתכנת המתחיל עלול להתקשות בהבנת תוכניות הכתובות בסגנון מקוצר, ורצוי לקחת זאת בחשבון.



שים לב!

משפטי הצבה מסוג זה

```
x= +y ;
x= -y ;
```

***אינם** כתיבה מקוצרת, אלא משפטי הצבה רגילים. המשפט הראשון מציב ל-x את ערכו של y והמשפט השני מציב ל-x את הערך ההפוך של y.*

תרגילים

תשובות בתקליטור: קבצים C:\tclite\targilim\c2e1-c2e9

הערה: שימו לב למשפטי ההצבה. בחלק ממשפטי ההצבה מעורבים משתנים מסוג int, float ו-char. את התוצאות במקרה בו מעורבים יותר מסוג משתנה אחד, אי אפשר לחזות. אנו ממליצים לכם להריץ את התרגילים ולראות מה התוצאה שמתקבלת.

1. לפניך משפטי הצבה. ציין את המשפטים השגויים:

א.
```
w = x+5*y  /* this is a remark */
```

ב.
```
p + m = r / (x + m);
```

ג.
```
l = (b % c) * d + r;
```

ד.
```
int i, j, k;
char c;
c = i * j * k + c;
```

ה.
```
x := r / 3.14 + 1000.053;
```

ו.
```
x /= 3.14;
```

2. עבור כל אחד מהמשפטים הבאים כתוב, מה יהיה ערכו של משתנה ההצבה (התייחס לטיפוסי המשתנים):

א.
```
int x, y;
char p ;
p = 'a'; y = 271;
x = y + p;
```

ב.
```
int y;
char x, p;
p = 'a'; y = 271;
x = y + p;
```

ג.
```
int x, y;
float f;
x = y = 13000;
f = x * y;
```

ד.
```
int x, y;
float f;
x = 13000; f = 13000.0;
y = x * f;
```

ה.
```
int x, y;
char chr;
float flt;
x = chr = 'b';
flt = (chr == x);
y = flt / x;
```

ו.
```
double dbl;
float f_num;
dbl = 13.289105;
f_num = 0.56;
dbl = --dbl * f_num ;
```

ז.
```
double dbl;
float f_num;
f_num = 0.56; dbl = 13.289105;
dbl -= f_num;
```

3. מה יהיה ערך משתני ההצבה בגמר פעולות החישוב הבאות. כתוב את הערכים של כל המשתנים בביטוי.

א.
```
int i, j, k;
i = j = 5;
k = i-- * j;
```

ב.
```
int i, j, k;
i = j = 5;
k= --i * j;
```

ג.
```
float f_num;
char byte;
byte = 255;
f_num = ++byte;
```

ד. שים לב שהתוצאות תלויות במהדר.

```
char c;
c = 5;
c = --c;
```

ה. שים לב שהתוצאות תלויות במהדר.

```
char c;
c = 5;
c = c--;
```

4. בדומה לדוגמאות שבפרק זה, כתוב קטע תוכנית המחשבת את העלות של השקיית גינה פרטית במשך שנה.

נתון:

- ❖ שטח הגינה: 1000-200 מ״ר (מספר ממשי).
- ❖ מחיר מטר מעוקב מים: 0.8 - 4.5 שקל.
- ❖ צריכת מים למ״ר למחזור השקיה: 0.2 - 0.3 מ״ק.
- ❖ מחזור השקיה מבוצע כל שלושה ימים.

בחר במשתנים מתאימים וכתוב משפטי הצבה עבור הנתונים הבאים:

- ❖ 325 מטר
- ❖ מחיר למ״ק: 2.5 ש״ח
- ❖ צריכת מים: 0.22 מ״ק

צרף לתוכנית הערות הסבר.

5. כתוב קטע תוכנית המחשבת ומציבה למשתנה tax_percent את אחוז המס הכולל המוטל על משכורת של 3600 שקל, אם ידועים שיעורי המס השולי הבאים:

עד 600 שקל - 0%

1000-600 שקל - 10%

1500-1000 שקל - 20%

1800-1500 שקל - 30%

2500-1800 שקל - 40%

3000-2500 שקל - 50%

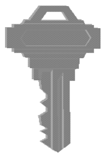
3500-3000 שקל - 55%

3500-4000 שקל - 60%

הקפד לבחור טיפוסים מתאימים ושמות משמעותיים למשתנים.

6. כתוב קטע תוכנית במספר מינימלי של משפטים, כדי לחשב היקף ושטח של מעגלים שרדיוסם: 4, 5, 6.

תוצאות החישובים יוצבו למשתנים: area1_3, length1_3 בהתאמה.



רמז!

השתמש במנגנון הקטנה/הגדלה עצמית.

7. נתונה הפונקציה getch() המחזירה תו. כתוב תוכנית הקולטת שלושה תווים ומציבה למשתנה מתאים את המנה:

תו_שני * תו_ראשון
―――――――――
תו_שלישי

שים לב!

הקריאה לפונקציה getch() נעשית כך:

```
char chr1;
chr1 = getch();
```

8. מה יהיו הערכים של x ו-y בתוכנית הבאה?

```
main()
{
int x, i, j;
char p, m;
float y;

i = j = 151;
p = 'b';
m = (p & Ox1F)   /* This is a note */
*i;
p *= i;
j += m;
x = j * p;
y = p / i;
}
```

9. מה יהיה ערכם של z, p ו-q בתוכנית הבאה?

```
main()
{
char z;
float p;
double q;

p = 5.7;
z = ++p;
z *= p--;
q = z * p--;
}
```

פרק 3

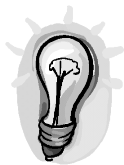
קלט ופלט באמצעות פונקציות ספריה

תוכניות המחשב מקיימות קשר במידה קטנה או גדולה עם העולם החיצון, כדי לקבל נתונים לעיבוד וכדי להציג את התוצאות עבור המשתמש. פעולות **קלט/פלט**, או **Input/Output** (I/O) מבוצעות באמצעי קלט/פלט רבים ושונים, שניתן לחבר למחשב. בהמשך נתייחס לפעולות הבסיסיות: קליטת תווים מה**מקלדת** (Keyboard) והצגת תוצאות על **המסך** (Display).

שפת C **אינה** מכילה פקודות קלט/פלט. שירותים אלה מסופקים באמצעות פונקציות ייחודיות מוכנות מראש, הנמצאות בספריה הסטנדרטית של מהדר C ואשר חלקן כתובות בשפות אחרות, כמו אסמבלר למשל (לצורך יעילות). ספריית המערכת מכילה, בנוסף לפונקציות I/O, עוד פונקציות רבות אחרות. במערכות מחשב ובמהדרים שונים מופיעה ספריה זו תחת שמות שונים. עליך לבדוק כיצד הספריה נקראת וכיצד להשתמש בפונקציות אלו בסביבת העבודה שלך במחשב (UNIX ,Windows ,DOS). נציין רק, שרוב המהדרים משתדלים לעמוד בתקן מקובל, ועל פיו נציג גם אנו את הפונקציות הבאות.

ברוב המהדרים ה**ספריה הסטנדרטית** נקראת **stdlib**, קיצור של Standard Library. תוכנית C המשתמשת בפונקציות I/O מהספריה, חייבת להכיל בתחילתה את השורה הבאה:

```
#include <stdio.h>
```



טיפ!

*כאן נציין רק, שהקובץ **stdio.h** (Standard input output library) מכיל הגדרות הנחוצות לשימוש בפונקציות I/O משפט #include "מודיע" למהדר להתייחס להגדרות אלו בתהליך ההידור.*

קליטת תו מהמקלדת

הפונקציה הבסיסית לקליטת תו בודד מהמקלדת נקראת **getch()**. כאשר היא מופעלת, היא ממתינה להקשה של תו כלשהו במקלדת, מקבלת ומעבירה אותו לתוכנית. התו מתקבל באופן מיידי ואין צורך בהקשה על מקש Enter. השימוש בפונקציה בתוך תוכנית נעשה באמצעות הצבה לתוך משתנה מטיפוס **char**.

בדוגמה הבאה פונקציית הקלט getch() תמתין להקשת תו במקלדת וערך ASCII שלו יוצב למשתנה ch.

```
#include <stdio.h>
#include <conio.h>
  ...
char  ch ;
  ...
ch = getch() ;
```

הקרנת תו בודד על המסך

הפונקציה המשלימה לקליטת תו מיועדת **להציג תו** בודד על המסך, וכינויה **putch()**. התו יופיע במקום בו נמצא באותו רגע **הסמן** (Cursor), אשר יתקדם במקום אחד ימינה בשורת המסך (כתיבה באנגלית). תצוגת התו תהיה כאשר נרשום בתוך הסוגריים את התו הרצוי ונפעיל את הפונקציה. התו יכול להיות קבוע מטיפוס תו, או שהוא יכול להימצא בתוך משתנה מטיפוס תו. ניתן להשתמש גם במשתנה מטיפוס אחר, בתנאי שהערך המצוי בו הוא בין 0 ל-255.

בדוגמה הבאה, התוכנית קולטת תו מהמשתמש, מציגה אותו על המסך ואחריו את התו "**!**".

```
        /*****      chp0301.c  :קובץ תקליטור        *****/
#include <stdio.h>
#include <conio.h>
char ch;
void main(void)
{
  ch = getch();
  putch (ch);
  putch ('!');
}
```

אם הקשת על מקש הנושא את הספרה 1 יופיע הפלט הבא, ובסיום יהיה הסמן מימין לסימן הקריאה:

```
1!
```

כדי לגרום לסמן לדלג אוטומטית לשורה חדשה, אפשר "להציג" את התו המיוחד המציין **שורה חדשה** (newline) - **n**. הפקודה המתאימה תיראה כך:

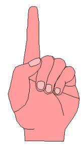
```
putch('\n') ;
```

באמצעות הכלים האלה ניתן לכתוב תוכנית C פשוטה, המציגה במסך את התווים המוקשים במקלדת. נקרא לתוכנית בשם **echo.c**, כי היא מזכירה תופעת "הד".

```
        /*****      chp0302.c :קובץ תקליטור       *****/
#include <stdio.h>
#include <conio.h>

void main(void)
{
char c;
  for (;;)
    {     /*לולאה אינסופית*/
    c = getch();
    putch(c);
    }
}
```

הפקודה **for** בתוכנית זו גורמת לצמד המשפטים שאחריה, בין הסוגריים המסולסלים, להתבצע בלולאה אינסופית. את מבנה משפט for והשימוש בו נלמד בפרק הבא.



שים לב!

1. *יש פונקציית ספריה נוספת הקולטת ומציגה בעצמה את התו על המסך, ושמה: **getchar()**.*
2. *לצורך ביצוע **תהליך הפוך** (דחיפת תו חזרה לאזור הקלט) נועדה הפונקציה **ungetch()**. תהליך זה מאפשר לקרוא שוב לתו על ידי הפונקציה getch().*

פלט לפי תבנית

שירותי הפונקציה putch() אינם מספיקים בדרך כלל, כאשר מדובר בפלט מורכב יותר. דרוש טיפול מיוחד בפלט המכיל מלל מעורב עם ערכים מספריים מתוך משתנים. לצורך זה קיימת הפונקציה **printf()** המטפלת בהפקת **פלט לפי תבנית** (Formatted Output).

התבנית הכללית של קריאה לפונקציה ()printf:

```
printf (<מחרוזת בקרה>, <ביטוי>, <ביטוי>...) ;
```

<**ביטוי**> קבוע, משתנה או ביטוי מורכב. מספר הביטויים אינו מוגבל.

<**מחרוזת בקרה**> מחרוזת תווים המכילה בתוכה תווי בקרה, שתפקידם לפענח את רשימת הביטויים. הפונקציה ()printf מתרגמת את ערכי הביטויים לפי סדר, על פי תווי הבקרה, ומציבה את התוצאות **במקום** תווי הבקרה. את המחרוזת הסופית, המורכבת מהתווים המקוריים של המחרוזת והערכים המשובצים בה, מציגה הפונקציה ()printf על המסך.

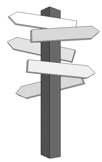
קטע מהתוכנית הבאה מדגים שילוב של ערכים בפקודת הדפסה:

```
int num1, num2 ;
...
num1 = 10 ;
num2 = 105 ;
printf("the sum of %d and %d is %d", num1, num2,
        num1+num2);
```

על המסך יופיע המשפט הבא:

```
the sum of 10 and  105 is 115
```

בדוגמה זו ניתן להבחין בכך שצירוף התווים **%d** מהווה **בקרה**, שהוחלף בערך של משתנה שלם. ההחלפה נעשתה לפי סדר המשתנים ברשימת הביטויים.



נקודות לציון!

א. *מחרוזת הבקרה מוגדרת בתוך זוג גרשים **שאינם מופיעים בפלט**.*

ב. *מספר תווי הבקרה **זהה** למספר הביטויים.*

ג. *מחרוזת הבקרה וסדרת הביטויים מופרדים באמצעות **פסיקים**.*

ד. ***כל תו** במחרוזת הבקרה, שאינו תו בקרה (בעל קידומת %) יוצג במחרוזת הסופית **כמו שהוא**.*

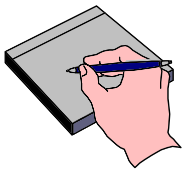
ה. *כדי להציג את התווים %, ", ', \, צריך להקדים בתו \, כלומר: \%, \", \', \\.*

תווי בקרה של הפונקציה ()printf

בטבלה 3.1 ניתנת רשימה חלקית של תווי הבקרה המותרים במחרוזת הבקרה של printf().

טבלה 3.1: *תווי בקרה של printf ()*

תוצאות הפענוח של הביטוי	משמעות	תו בקרה
ייצוג עשרוני של שלם	decimal	%d
תו בודד בעל קוד ASCII של ערך הביטוי	character	%c
מחרוזת תווים. כל התווים במחרוזת יוצגו עד לתו NULL (ASCII 0)	string	%s
ייצוג הקסדצימלי (בסיס 16)	hexadecimal	%x
ייצוג עשרוני ללא סימן (שלם)	unsigned	%u
ייצוג אוקטאלי (בסיס 8)	octal	%o
ייצוג עשרוני הכולל נקודה עשרונית ושבר (חוקי עבור משתנים/קבועים מטיפוס float/double)	float	%f
ייצוג עשרוני, כולל נקודה עשרונית וחזקה של 10 הצמודה למספר מימין עם קידומת E (1.24E-5)	exponent	%e



הערה!

*נדגיש, כי הטבלה הזו מכילה את תווי הבקרה **הנפוצים והמקובלים ביותר**. לכל מהדר יש תווי בקרה נוספים, אשר מאפשרים פעולות נוספות להצגת הפלט.*

לדוגמה, נתונה תוכנית:

```
/*****      קובץ תקליטור: chp0303.c      *****/

#include <stdio.h>

char ch;
int i;
float f;
char s[22]="A STRING DEFINED IN C";

void main (void)
{
   ch = 'a';
   i = f = 155;
   f = f/7;
```

```
printf("i decimal - %d ,i hexa - %x\n",i,i);
printf("ch contains : %c, ascii code is %d\n", ch, ch);
printf("#%s#\n", s);
printf("regular float - %f,exponent - %e\n", f, f);
printf("line 1 sum = %d \n line 2 contains %s \n", i+100, s);
printf("The value of i is %d", i);
printf("and the sum of c %x \n", 'a');
printf("i = >%10d<\n f = *%10.3f*", i, f);
}
```

נציג את הפלט לאחר ביצוע פקודות printf שונות :

א.
```
printf("i decimal - %d, i hexa - %x\n", i, i);
```

הפלט :
```
i decimal -  155, i hexa - 9b
```

ב.
```
printf("ch contains : %c, ascii code is %d\n", ch, ch);
```

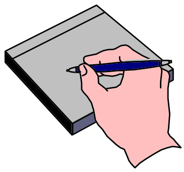
הפלט :
```
ch contains:  a, ascii code is 97
```

ג.
```
printf("#%s#\n", s);
```

הפלט :
```
# A STRING    DEFINED IN C#
```

ד.
```
printf("regular float - %f, exponent - %e\n", f, f);
```

הפלט :
```
regular float  -22.142857,exponent-2.214286E+001
```



הערה!

*בכל הדוגמאות לעיל, הפלט מופיע על המסך ברצף, ללא דילוג לשורה חדשה. כדי לעבור לשורה חדשה צריך לכלול את התו "**\n**" בפקודה printf.*

באמצעות פקודה printf בודדת ניתן להציג על המסך מספר משפטים, אשר כל אחד מהם מתחיל בשורה חדשה :

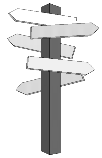
```
printf("line 1 sum = %d \n line 2 contains %s \n", I=255, s);
```

הפלט :
```
line 1 sum = 255
line 2 contains A  STRING DEFINED IN C
```

תוצאה שונה נקבל כאשר נקרא פעמיים לפונקציה ()printf, מבלי להשתמש בסימן "שורה חדשה" בפקודת printf הראשונה. למשל :

```
printf("The value of i is %d", i);
printf("and the sum of c is %x \n", 'a');
```

הפלט :
```
The value of i is  155 and the sum of c is 61
```



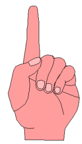
נקודות לציון!

א. ניתן לפענח את אותו ביטוי/משתנה באמצעות תווי בקרה שונים, וכך לשנות את צורת הפלט.

ב. <ביטוי> יכול להיות קבוע, משתנה, או פעולה אריתמטית כלשהי. בהמשך נראה ש- <ביטוי> יכול להיות גם קריאה לפונקציה.

ג. כל תווי הבקרה מתחילים בסימן %.

ד. תווי הבקרה במחרוזת הבקרה, תופסים בדרך כלל 2 מקומות בלבד. לכן, לאחר שיבוץ הביטויים, מחרוזת הפלט תהיה בדרך כלל ארוכה יותר.



שים לב!

ניתן לשבור שורת ;()printf על ידי התו '\' במידה והמחרוזת ארוכה מדי.

דוגמה:

```
printf("The value of :\ is %d", i);
```

רוחב שדה הפלט

בנוסף לתווי הבקרה הבסיסיים שהוצגו בסעיף הקודם, יש מספר תווים **אופציונליים** נוספים, המאפשרים שליטה טובה יותר על צורת הפלט. לשם כך צריך להשתמש במציין רוחב שדה הפלט. **מציין** זה הינו מספר עשרוני המופיע בין סימן "%" (אחוז) לבין תו הבקרה וצורתו הכללית היא: **w.p** כאשר, **w** מגדיר את **רוחב השדה** (Width), ו-**p** מגדיר את **דיוק השדה** (Precision), למשל: %7d, %8.3f.

אופן הפענוח של המציין תלוי בסוג תו הבקרה, כפי שתראה להלן.

רוחב השדה (w)

עבור תווי הבקרה **o, u, x, c, d** תופיע התוצאה **לפחות** ברוחב w. אם יש פחות תווים מאשר השדה יכול להכיל - יתוספו רווחים משמאל. אם יש יותר תווים - הם יופיעו במלואם, ויגלשו מרוחב השדה w (שים לב למצב זה!).

עבור תווי הבקרה **e, f** יופיע המספר העשרוני (כולל השבר והנקודה העשרונית) **לפחות** ברוחב w. אם יש פחות תווים - יתוספו רווחים משמאל. אם יש יותר תווים - הם יופיעו במלואם, ויגלשו מהרוחב w.

עבור תו הבקרה **s** תופיע המחרוזת **לפחות** ברוחב w. גם כאן, כבמקרים קודמים, יתוספו רווחים משמאל, או שתהיה גלישה, בהתאם לאורך המחרוזת בפועל לעומת רוחב השדה הנתון w.



הערה!

*יש מהדרים שמאפשרים למלא **אפסים מובילים** (leading zeros) במקום רווחים.*

דיוק השדה (p)

עבור תווי הבקרה **e**, **f** מציין p את מספר הספרות שיש להציג לאחר הנקודה העשרונית.

עבור תו הבקרה **s** מציין p את המספר המקסימלי של תווים שיש להציג.

דוגמאות:

א. התוכנית (תו בקרה d ומציין w):

```
int  i ;
i = 155 ;
printf ("i = >%10d<  >%5d<  >%2d< \n", i, i, i);
```

הפלט המתקבל:

```
i = >     155<  >  155<  >155<
```

ב. התוכנית (תו בקרה f ומציינים w.p):

```
float  f ;
f = 103.4732 ;
printf ("f = >%10.2f<  >%8.4f<  >%5.0f<\n",f,f,f,);
```

הפלט המתקבל:

```
f = >     103.47<  >103.4732<  > 103<
```

ג. התוכנית (תו בקרה s ומציינים w ו-w.p):

```
printf ("#%10s#  #%5.3s#\n", "C-string", "C-string");
```

הפלט המתקבל:

```
#  C-string#  #    C-s#
```

קלט לפי תבנית

בדומה לפונקציה ()printf המטפלת בפלט לפי תבנית, יש פונקציית ספריה מקבילה המטפלת בקלט לפי תבנית. פונקציה זו נקראת **()scanf**.

הצורה הכללית של קריאה לפונקציה ()scanf :

scanf(<מחרוזת בקרה>, <כתובת משתנה>, <כתובת משתנה>,...) **;**

<**מחרוזת בקרה**> מחרוזת של תווים בדומה למחרוזת הבקרה שבפונקציה ()printf.

<**כתובת משתנה**> הגדרת תא הזיכרון שאליו יוכנס ערך הקלט המתאים.

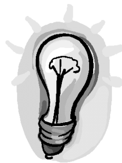
בשלב זה לא נסביר את משמעות המושג "**כתובת משתנה**". הסבר מלא לכך יינתן בפרק 11 העוסק במצביעים.

איך פועלת הפונקציה? הפונקציה ()scanf סורקת את מחרוזת הבקרה, וקולטת ערכים לתוך משתנים. הפונקציה מבצעת את ההמרות המתאימות מתווי הקלט לטיפוסים המוגדרים במחרוזת הבקרה. ההפרדה בין ערכי הקלט נעשית באמצעות רווחים.

תרגילים

תשובות בתקליטור : קבצים C:\tclite\targilim\c3e1-c3e6

1. כתוב תוכנית שתקבל כקלט מספר בן שלוש ספרות המייצג שטח מעגל, והדפס את הרדיוס שלו.



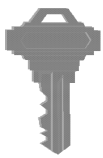
הנחיה!

קלוט את הספרות זו אחר זו וחשב את הערך שלהן, על ידי כפל ב-10. השתמש במספר מינימלי של משתנים. הדפס את התוצאה בעזרת הפונקציה ()printf.

2. כתוב משפט, אשר ידפיס את המספר 123.572 בשדה שרוחבו הכולל 6 תווים עם שתי ספרות עשרוניות.

3. כתוב קטע תוכנית שתקלוט תו ותדפיס את המשפט הבא במבנה המתואר להלן :

```
The value
   of <תו> is <ASCII ערך>
and this is the end.
```



רמז!

הקוד הגורם למעבר לשורה חדשה הוא בן שני תווים, שערך ASCII שלהם OxOA.

4. איך יוצג פלט ההדפסה של המשפט הבא?

```
printf("This sentence prints %5.3e %c \n!", 155.3361,
     ox0A);
```

5. איך יוצג פלט ההדפסה של המשפט הבא?

```
printf("1.\n %s, 2.\n %s, %c \n", "First line",
     "sec\nond line", '?');
```

6. שכתב את תוכנית 5 שכתבת בפרק 2, כדי שתקלוט את גובה המשכורת (ארבע ספרות) ותדפיס את אחוז המס הכולל.

פרק 4

משפטי בקרה בשפת C

בדומה לשפות מחשב אחרות, מציעה שפת C מיגוון של **משפטי בקרה** כדי לאפשר **כתיבת תוכנית יעילה, חסכונית וברורה**. משפטי הבקרה נגזרים מאופיים של קטעי התוכנית השונים: קטעי תוכנית אחדים יבוצעו רק כאשר יתקיימו תנאים מסוימים, ואחרים יבוצעו פעם אחר פעם כל עוד מתקיימים תנאים מסוימים.

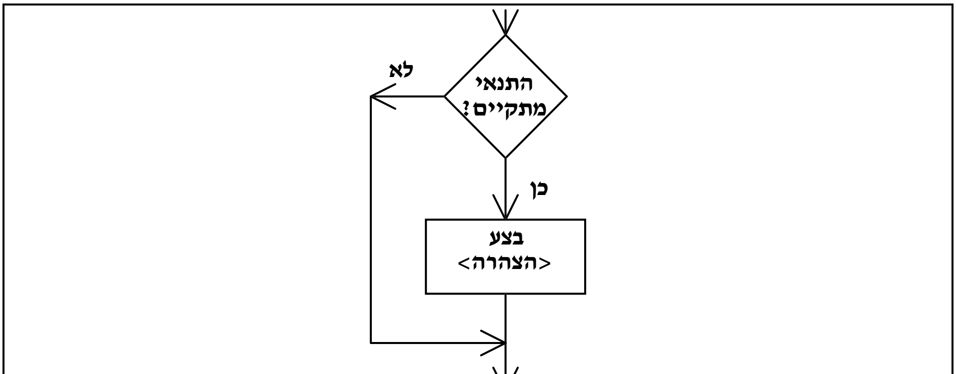
בפרק זה נלמד להכיר את משפטי הבקרה הבסיסיים בשפת C, נבחן דוגמאות לכל אחד מהם ונכיר כלים חשובים ל**כתיבת תוכניות יעילות וקריאות**.

משפט התנאי - IF

מדוע נחוץ משפט IF ?

נניח שברצונך לבצע מספר פעולות - **אם ורק אם** מתקיימים תנאים מסוימים. לדוגמה, ברצונך לחלק את המספר 10000 במשתנה x, אך אינך יודע את ערכו ואינך רוצה מסיבות מובנות לעשות זאת, אם ערכו יהיה 0. במילים אחרות: אם x אינו שווה ל-0 בצע 10000/x.

את המקרה הכללי ניתן לתאר בתרשים הבא:



תרשים 4.1: if

התחביר של משפט IF:

```
if (<תנאי>)   <הצהרה>  ;
```

נחזור לדוגמה הקודמת:

```
int x;
float result ;
. . .
if (x!=0)
  result = 10000/x ;
```

ניתן לראות, ש**ההצהרה** נכתבה **בשורה נפרדת**. התחביר אינו מחייב זאת, אולם אם ברצונך לשפר את קריאות התוכנית, אנו ממליצים על דרך כתיבה זו. כפי שראינו קודם, <תנאי> יכול להיות גם ביטוי מורכב.

כדי ש- <הצהרה> תכלול קבוצה של משפטים (זהו "בלוק"), נשתמש בסוגריים מסולסלים כדי להקיף אותם. לדוגמה,

```
int x, y ;
float result = 0;
. . .
if (x != o && result == 0)
  {
    result = 10000 / x ;
    x += y ;
    --y ;
  }
```

סדרת המשפטים, שבין הסוגריים המסולסלים תתבצע **אם ורק אם**, ערכו של x שונה מ-0, **וגם** ערך result שווה ל-0. כל סדרת משפטים בשפת C שנמצאים בין סוגריים מסולסלים, נחשבים **למשפט הצהרה מורכב**. מכאן אפשר להסיק, שבתיאור התחבירי של משפט if הבא

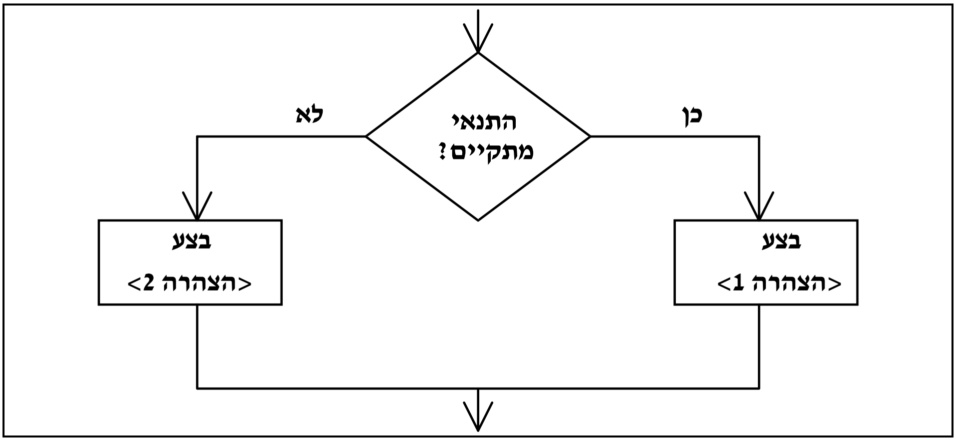
```
if (<תנאי>)   <הצהרה>  ;
```

הכוונה ב-<הצהרה> למשפט בודד, או למשפט מורכב (דהיינו, סדרת משפטים מתוחמים בסוגריים מסולסלים). כלל זה מתקיים בכל משפטי הבקרה שנלמד בהמשך.

IF..ELSE

משפט הבקרה if..else מאפשר בחירה בין שני קטעי תוכנית שונים, בהתאם לערכו של תנאי מסוים.

התרשים המתאים למבנה זה:



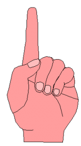
תרשים 4.2: if.. else

התחביר של משפט if..else:

```
if (<תנאי>)   <הצהרה 1>   else <הצהרה 2>  ;
```

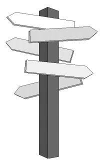
נניח, על פי הדוגמה הקודמת, שברצוננו להתריע כאשר המשתנה x קיבל ערך 0, ולכן לא בוצעה חלוקה. קטע התוכנית המתאים יהיה:

```
if (x != 0)
   result = 10000 / x ;
else
   printf("x equals 0.Cannot perform division !! \n");
```



שים לב לצורת הכתיבה!

*ההצהרות "מוזזות" ימינה והמילים if-else נמצאות **זו מעל זו**. צורת כתיבה זו נוחה מאוד כאשר תכתוב תוכנית שלמה, מכיון שהיא מאפשרת לזהות את הקשרים שבין הקטעים השונים, ולאיזה if "שייכת" כל מילת else. עם זאת, לשפת C כל זאת לא איכפת!*



כאן המקום לציין שבשפת C מתקיים הכלל הבא:

*כל **else** "שייך" ל-**if** האחרון שאינו "סגור" על ידי מילת else, ושאינו נמצא בתוך סוגריים מסולסלים.*

הסתכל בדוגמה הבאה:

```
if (result == 0)
   if (x != 0)
      result = 10000 / x ;
   else
      printf("x equals 0. Cannot perform ... \n") ;
```

משפט זה דומה במובנו למשפט המורכב שראינו קודם, אך הוא הושג על ידי שרשור, או **קינון** (nesting) של **שני** משפטי if. ניתן גם לראות ש-else מתייחס לתנאי השני. אם נקבל תוצאה - result - שונה מ-0, לא נבדוק ולא נדע אם הוא שונה מ-0. אם ברצונך **לוודא ש-else יתייחס ל-if מסוים**, השתמש בסוגריים מסולסלים, כדי לתחום את משפט if הרצוי.

דוגמה:

```
if (result ==0)
   {
     if (x != 0)
         result = 10000 / x ;
   }
else
   printf("result is not zero \n ") ;
```

כאן המקום להזהיר מפני טעות נפוצה הנגרמת כתוצאה ממשאלת לב. הקטע הבא יובן על ידי המהדר **בצורה שונה** מכוונתו המקורית של הכותב:

```
if (result ==0)
   if (x != 0)
     {
         result = 10000 / x ;
     }
else
   printf("result is not zero \n ") ;
```

else במקרה זה "שייך" ל-if השני, למרות שהוא כתוב מתחת לראשון, בהתאם לכלל שהוסבר למעלה (כזכור, לשפת C לא חשובה צורת הכתיבה/עיצוב של הפקודות, אלא התוכן שלהן).

בעזרת מנגנון if-else ניתן לתכנת **מבנה "רב אפשרויות"** (N-Way Choice).

בכל שלב נבדוק קיום של תנאי מסוים, ובעזרת else נאפשר את המשך הבדיקות אם התנאי אינו מתקיים. לדוגמה, נציג קטע תוכנית הבודק אם תו מסוים הוא אות גדולה, אות קטנה, מספר, או סימן אחר.

```
         /*****     chp0401.c :קובץ תקליטור      *****/
#include <stdio.h>
#include <conio.h>
char c;
void main(void)
{
   c = getch();
   if (c >= 'a' && c <= 'z')
     printf("small letter \n");
   else if (c >= 'A' && c <= 'Z')
     printf("big letter \n");
```

```
   else if (c >= '0' && c <= '9')
     printf (" number \n");
   else
     printf(" other character \n");
}
```

ניתן לראות, שכאן חרגנו מכללי הכתיבה הקודמים כדי להדגיש את אופיו המיוחד של הקטע. כלומר, לא כתבנו כל else תחת if המתאים לו. חריגה זו נהוגה רק במקרה מיוחד זה.

משפטי if מיוחדים

למדנו עד כה שכאשר ערכו של ביטוי התנאי (במסגרת if) שונה מ-0, הוא נחשב "אמת" (True). אם כך, אזי המשפט הבא יהיה **אמת**, כל עוד המשתנה num שונה מאפס, לדוגמה,

```
int num=8;
```

במקום כל "משתנה" הרשום בסוגריים של משפט התנאי, אפשר לכתוב משפט אחד, או "**בלוק**" - קבוצה של משפטים.

```
if(num)
{
   משפט/בלוק
}
```

לכן, **אין צורך** לשאול if(num!=0) מכיון שמנגנון השפה בודק זאת בעצמו.

כיצד אם כן, נבצע את משפט if, אם ברצוננו לבצעו **רק** כאשר התנאי הוא 0.

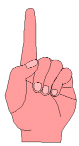
```
int num=0;
if (!num)
   משפט/בלוק
```

השאלה הנשאלת עתה היא האם **לא שונה** מ-0 פרושו: לא, התנאי לא שונה מ-0, הוא שווה ל- 0 ולכן יתבצע משפט if.

ניתן לבצע משפט השמה במסגרת תנאי if, למרות אזהרת המהדר, ה"מבין" כשיש סימן שוויון אחד בלבד זוהי פעולת השמה ולא פעולת בדיקת שוויון בתוך תנאי.

```
int a=3, b=0;
if (a=b)
```

תוצאת משפט if היא "שקר" (false).



שים לב!

לא מקובל להשתמש בביטוי תנאי מסוג זה.

לולאות

כוחו של מחשב וכוחה של שפת תכנות באים לידי ביטוי במספר תחומים עיקריים. אחד מהם הן **לולאות** (Loops). הלולאה מאפשרת לבצע פעולה נשנית מספר פעמים רב, תוך שמירת היכולת לבקר ולנתח נתונים במהלך הביצוע. ככל שתמיכת שפת התכנות בלולאות רבה יותר - נוכל ביתר קלות לנצל את יתרונות הלולאה בבניית תוכניות יעילות יותר, המתבססות על קטעים שחוזרים על עצמם.

כפי שנראה, מאפשרת שפת C משפטי בקרה נוחים וקומפקטיים, העונים למיגוון צרכים. בחלק מהמקרים ניתן לפתור אותה בעיה בעזרת משפטי בקרה אחדים, אולם במרביתם קיימת העדפה למשפט מסוים - המתאים והיעיל ביותר עבור אותו מקרה.

שפת C תומכת בשלושה סוגים שונים של משפטי לולאה.

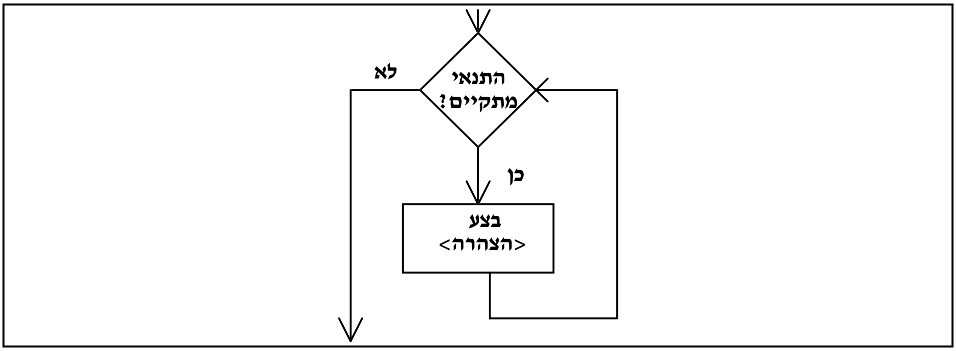
משפט WHILE

בהקדמה לפרק זה הזכרנו את משפטי הבקרה העיקריים. מבנה **משפט While** מתאים למקרה "**כל עוד מתקיים התנאי - בצע...**".

תחביר המשפט:

```
while (<תנאי>)   <הצהרה> ;
```

התרשים:



***תרשים 4.3:** תרשים while*

כפי שניתן לראות, התנאי נבדק **לפני** כל ביצוע של <**הצהרה**>.

מתי נשתמש במשפט זה?

כאשר ברצוננו להבטיח ש- <הצהרה> תתבצע רק כל עוד <תנאי> מתקיים. במצבים מסוימים ייתכן ש- <הצהרה> לא תתבצע כלל:

```
i = 0 ;
while (i != 0)
   printf(" i = %d \n ", i) ;
```

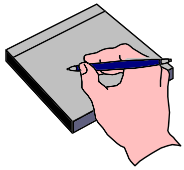
בדוגמה זו, משפט ()printf לא יבוצע כלל.

דוגמה:

נניח שברצוננו לכתוב את הפונקציה !n (n עצרת) עבור n = 7.

```
int n, product ;
...
n = product = 1 ;
while (n <= 7)
  {
    product = product * n ;
    n = n + 1 ;
  }
```

בקטע זה ביצענו 7 פעולות כפל, וספרנו באמצעות המשתנה n את מספר הפעמים של ביצוע הלולאה. לכן, אנו מכנים את **n** בשם "**משתנה בקרת הלולאה**".



הערה!

*כאן המקום להזהיר מפני **טעות נפוצה** בבניית לולאות, במיוחד באלו המכילות קטע תוכנית ארוך: קל **לשכוח** "פרט פעוט", כמו **עדכון של משתנה בקרת הלולאה**. אם לא נבקר כראוי את מהלך ביצוע הלולאה, היא תתבצע עד אינסוף. בדוגמה הקודמת, הסרת המשפט **n = n +1** תגרום לתוצאה כזאת.*

לסיום, נשתמש ב"קיצורי דרך" שמאפשרת שפת C כדי לצמצם את מספר הפקודות בתוכנית שלמעלה. כך נפיק תוכנית קריאה יותר וגם נשלטת יותר על ידנו, מכיון שתהיינה בה פחות פקודות.

```
        /*****      chp0402.c :קובץ תקליטור      *****/

#include <stdio.h>

int n=0, product=1;

void main(void)
{
  while (++n <=7)
    product *= n;

  printf("product of 7 (7!) is %d\n",product);

}
```



שאלה!

האם ניתן לכתוב קטע תוכנית זו בדרך שונה?

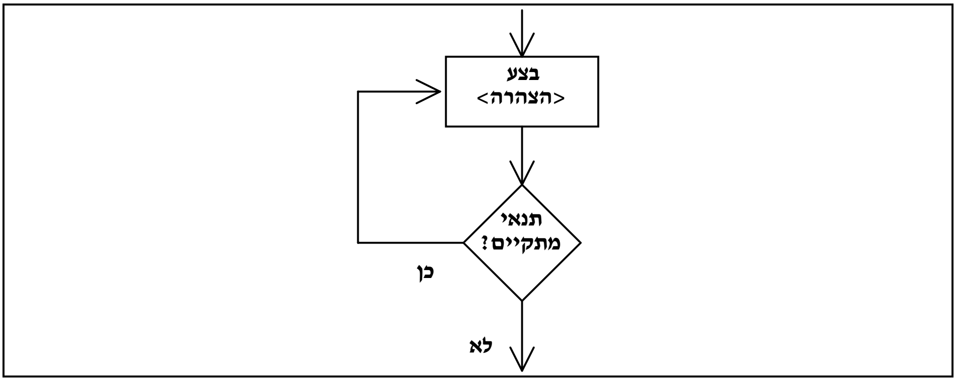
חשוב על pre/post increment.

משפט DO-WHILE

במקרים מסוימים יש לבצע את <**הצהרה**> במשפט הבקרה **לפחות** פעם אחת. כלומר, ראשית נבצע את <הצהרה> ורק בסופה נבדוק את <תנאי> ונחליט **אם** לבצעה שוב. זהו המצב ב**לולאות do-while**, שבה <תנאי> נבדק **בסוף** המשפט, **ורק** אם הוא מתקיים, תתבצע הלולאה פעם נוספת.

תחביר המשפט:

```
do <הצהרה>  while  (<תנאי>) ;
```



תרשים 4.4: *תרשים do-while*

את קטע התוכנית שבדוגמה הקודמת נוכל לכתוב כך:

```
        /*****     chp0403.c :קובץ תקליטור     *****/

#include <stdio.h>
int n, product;
void main(void)
{
     n = product = 1 ;
     do
        product *= n ;
     while (++n <=7) ;
     printf("product of 7 (7!) is %d\n",product);
}
```

דוגמה נוספת:

```
char key ;
do
 {
     key = getch()
     if (key == 'Y') .... ;
     else if (key == 'P') ... ;
     else if (key == 'A') ... ;
 }
while (key != 'E') ;
```

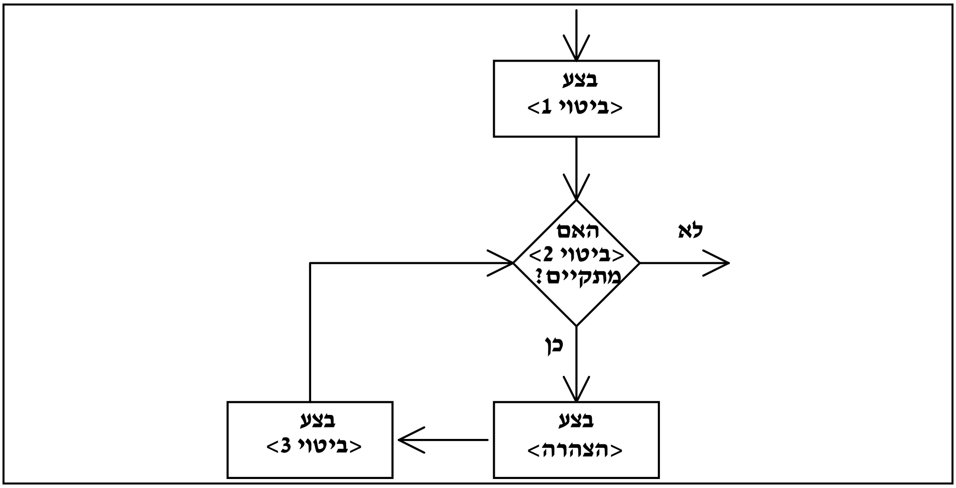
משפט FOR

משפט הבקרה הנפוץ והנוח ביותר בשפת C הוא **לולאת for**. לולאת for מכילה באורח קומפקטי ובשורה אחת את **שלושת השלבים הבסיסיים** של פעולות הלולאה, אשר עוסקים במשתנה בקרת הלולאה: אתחול, בדיקה של <תנאי> להמשך ביצוע, וקידום (או שינוי) של המשתנה. בהמשך הלימוד נכיר שימושים ויישומים רבים המנצלים לולאה חשובה זו. כעת נסתפק בהיכרות בסיסית ובמספר דוגמאות שיאפשרו לנו להעריך את אפשרויותיה.

תחביר המשפט:

```
for  (<ביטוי 1> ; <ביטוי 2> ; <ביטוי 3>)
          <הצהרה> ;
```

התרשים:



תרשים 4.5: *תרשים for*

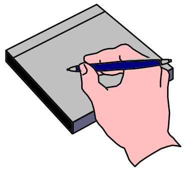
מהתרשים ניתן לראות, שלמעשה **כל לולאת for ניתנת להמרה בלולאת while**. לפני לולאת while מתבצע <ביטוי 1>, אח״כ מתבצעת ההצהרה, ולבסוף מתבצע <ביטוי 3>.

נתבונן בקטע התוכנית הבא:

```
for (exp1 ; exp2 ; exp3)
   {
         stmnt1 ;
         stmnt2 ;
   }
```

עיין בקטע התוכנית הבא:

```
exp1 ;
while (exp2)
   {
         stmnt1 ;
         stmnt2 ;
         exp3 ;
   }
```



הערה!

קטע זה שקול לקטע הקודם (for).

כיצד נתכנת את החישוב של **!7**?

```
int i, product = 1 ;
...
for (i=1 ; i<= 7 ; ++i)
     product *= i;
```

נעיין בשלושת הביטויים של משפט for:

בביטוי הראשון אתחלנו את משתנה בקרת הלולאה; בביטוי השני השווינו אותו לערך הגבול 7; ובביטוי השלישי קידמנו את i.

אפילו בדוגמה פשוטה זו ניתן לראות את היתרון בקומפקטיות והקריאות של הלולאה. ניתן לצמצם את הקטע הקודם עוד יותר:

```
int i, product ;
for (i = 1, product = 1 ; i <= 7 ; product *= i++) ;
```

הכרנו כאן שני גורמים חדשים:

א. **האופרטור** "," (פסיק) - מאפשר לבצע מספר פעולות עוקבות ולהישאר במסגרת של <**ביטוי 1**>.

   למעשה, אין הגבלה על מספר הפסיקים (המגבלה היא נוחות העבודה בלבד) וניתן לבצע מספר רב של פעולות במסגרת האתחול של המשתנים.

   כאן המקום להדגיש, שהשימוש באופרטור פסיק (",") אינו מוגבל ללולאת for בלבד, אלא זהו שימוש מקובל במשפטים שונים.

ב. **ההצהרה הריקה** - במקרה זה, כל הפעולות הדרושות מתבצעות במסגרת שלושת הביטויים שמופיעים במשפט for, ולכן אין צורך ב<הצהרה>. נדגיש עם זאת, ש**חובה לכתוב "**;**"** (נקודה-פסיק), המסיים את המשפט.

השימוש ב**הגדלה מאוחרת** (post increment) נועד לאפשר קידום של משתנה הלולאה בתוך פעולת הכפל, מבלי לפגוע בסדר הנכון של הביצוע; דהיינו, ביצוע המכפלה ורק אחר כך הגדלת המונה.



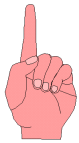
שאלה!

מה היה "קורה" לו הקטע נכתב כך:

```
         /*****      chp0404.c :קובץ תקליטור      *****/

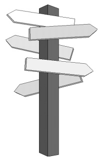
#include <stdio.h>
int n, product;
void main(void)
{
     for (n=product=1; n <= 7; product *= n++) ;

     printf("product of 7 (7!) is %d\n",product);
}
```



שים לב!

*גמישותה של לולאת for עלולה לגרור אותנו ל"**ניצול יתר**" של אפשרויותיה, עד כדי כך שהיא לא תהיה קריאה. בדוגמה הקודמת, גרמה הכנסת פעולת הכפל ל- <ביטוי 3> לקושי בקריאות ובהבנת הקטע. אנו ממליצים **להשתמש בלולאה זו בתשומת לב רבה** ולהימנע מלכלול בתוך הביטויים משפטים שאין להם קשר ישיר עם משתנה בקרת הלולאה.*



נקודת ציון!

*ראוי לציין יישום חשוב ועקרוני בלולאת for - **הביטוי הריק** (Null Statement). **כל אחד** משלושת הביטויים של משפט for יכול להיות ביטוי ריק.*

לדוגמה, אם <**ביטוי 1**> ריק, לא תתבצע פעולת האתחול.

```
int i = 1 ;
for (; i <=100 ; ++i)
     printf(" VALUE OF i IS : %d \n ", i) ;
```

אם <**ביטוי 3**> ריק, לא תתבצע פעולת העדכון במסגרת הביטויים האלה. אין זה מונע שהעדכון יתבצע בשלב אחר, אך אם לא דאגת לכך, תתקבל במרבית המקרים **לולאה אינסופית**. בדוגמה הבאה ניתן לראות שפעולת הקידום נעשתה במסגרת <**הצהרה**>.

```
int i ;
for (i = 1 ; i <= 100 ;)
     printf("THE VALUE OF i IS : %d \n ", i++) ;
```

מקרה מיוחד הוא, כאשר <**ביטוי 2**> ריק. כתוצאה נקבל לולאה אינסופית, מכיון שתנאי העצירה אינו קיים.

נכתוב עתה תוכנית, הקולטת תווים מן המקלדת ומציגה אותם על המסך. לדוגמה:

```
        /*****      chp0405.c  :קובץ תקליטור       *****/
#include <stdio.h>
#include <conio.h>
void main(void)
{
  char c;

  c = getch();
  for (;;c=getch() )
    putch(c);
}
```

מכיון ש<**ביטוי 2**> חסר, נקבל לולאה אינסופית. כדי להפסיקה עלינו להקיש Ctrl+C.

קינון משפטי for

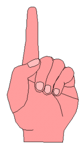
ניתן **לקנן** (nest) מספר רמות של משפטי for, זו בתוך זו, כמו בדוגמה הבאה:

```
for (...)
  {משפט/בלוק
    for (...)
       {
        משפט/בלוק
       }
    משפט/בלוק...
  }
```

הדבר דומה לסיבוב מחוגי השעון. משפט for העליון מייצג את השעות, והתחתון - את הדקות.

לדוגמה, להצגת מהלך יום, שבו 24 שעות, 60 דקות בשעה נכתוב את התוכנית הבאה:

```
int i,j;
for(i=o; i<24, i++)           i- מייצג את מחוג השעות
  for(j=0; j<60; j++)         j- מייצג את מחוג הדקות
```



שים לב!

לא מקובל לקנן יותר מאשר שלוש רמות (הסיבה היא אנושית ולא "מחשבית").

משפטי BREAK ,CONTINUE ,GOTO

עד כה למדנו כיצד להשתמש בלולאות השונות. אך כאשר נכנסנו כבר לגוף הלולאה - האם "נגזר" עלינו להישאר בה ולבצעה עד שתנאי העצירה יתקיים?

שפת C מאפשרת לבצע חלק מגוף הלולאה, או לצאת ממנה גם כאשר תנאי הסיום לא התמלא עדיין. קיימים לצורך זה שלושה משפטים, אשר נלמד להלן.

משפט BREAK

השימוש במשפט זה הוא הנפוץ ביותר, ומשמעותו - **הפסק**. כאשר במהלך ביצוע התוכנית נגיע למשפט **break**, הוא יגרום ליציאה מתוך משפט הבקרה שבו הוא נמצא ולהפסקת הלולאה. טווח ההשפעה שלו הוא **משפט הבקרה האחרון**, אשר בתוכו הוא נמצא. כאשר בתוכנית יש מספר משפטי בקרה זה בתוך זה, יגרום משפט break ליציאה ממשפט הבקרה שבו הוא נמצא, אל המשפט המכיל אותו (**פרט** למשפטי if..else). לדוגמה, המספר 1 כתנאי של לולאת while גורם לכך, שזו תהיה לולאה אינסופית, כי ערכו של 1 תמיד True. רק משפט break יוכל לחלץ אותנו מלולאה כזו.

```
          /*****      chp0406.c :קובץ תקליטור       *****/
#include <stdio.h>
#include <conio.h>
void main(void)
{
   char c;

   while (1)
      {
          c = getch();
          if (c == 'e')
             break;
      }
   printf("BREAK יציאה מלולאה אינסופית בעזרת פקודת\n");
}
```

מומלץ להימנע ממשפט break, ולצאת מלולאה בצורה מסודרת.

משפטי הבקרה המושפעים ממשפט break הינם: while, do-while, for ו-switch.

משפט CONTINUE

לא תמיד נרצה לצאת מהלולאה. לפעמים נחליט, שבתנאים מסוימים נבצע רק חלק ממנה. המשפט **continue** גורם להפסקת הביצוע של <**הצהרה**> במקום בו הוא מופיע, ודילוג לתחילת הלולאה לביצוע מהלך נוסף. השימוש במשפט continue נדיר, ולרוב גם מיותר, אבל לפעמים הוא נחוץ. נדגים שתי דרכים לתכנת לולאה עם המשפט continue ובלעדיו.

```
for (; ;)                          for (; ;)
  {                                  {
   ...                               ...
   if (x < 14)                       if (!(x <14))
     continue;                         {
   ...                                  ...
  }                                    }
                                     }
```

משפט GOTO

לאלה שמכירים שפות מכונה כמו אסמבלי, או את שפת BASIC, לא ייראה משפט זה יוצא דופן, כי הוא מקביל בתפקודו למשפטי JMP ו-GOTO בהתאמה. אולם בשפה מבנית ועילית כמו שפת C, או פסקל, זהו משפט שפרט להסבר אודותיו - הס מלהזכירו!

משתמשים ב-goto כאשר צריך לדלג למקום מסוים בתוכנית, שמציינים אותו תחילה ב**תווית** (Label). במקום שבו רוצים לדלג אל קטע התוכנית המתחיל בתווית זו, יש לכתוב:

goto <תווית>;

לדוגמה:

```
        /*****      chp0407.c :קובץ תקליטור       *****/
#include <stdio.h>
#include <conio.h>
void main(void)
{
   char c;
   while (1)
      {
      c = getch();
      if (c == 'e')
          goto FINISH;
      }
   printf("!!! לכאן לא נגיע לעולם\n");
FINISH:
   printf("GOTO יציאה מלולאה אינסופית בעזרת הפקודה\n");
}
```

התווית עצמה יכולה להימצא לפני או אחרי משפט goto שמפנה אליה. יש להצמיד לימין התווית סימן נקודתיים (":"). שים לב שהמילה "FINISH" נכתבה כך לשם הבלטה בלבד; ניתן לבחור תווית כלשהי!

מהדרים רבים מגבילים את טווחי הדילוג של משפט goto. על ידי תכנון נכון של מבנה התוכנית והפונקציות השונות תמצא שברוב המקרים **אין צורך** להשתמש במשפט זה.

משפט SWITCH

בסעיף קודם ראינו כיצד ניתן לממש **מבנה רב-אפשרויות** בעזרת משפט הבקרה המשולב IF-ELSE. במקרים שבהם ניתן לשייך כל אפשרות למספר שלם, מציעה שפת C משפט בקרה נוח ומצומצם יותר. נניח שברצוננו לבצע פעולות שונות כתלות בקלט מסדרת המשפטים שבתוכנית הבאה. במנגנון **..if** או **if..else** היינו מתכנתים זאת כך :

```
        /*****      chp0408.c  :קובץ תקליטור      *****/
#include <stdio.h>
void main(void)
{
  int count=8, total_sum=0 ;
  char key ;
  printf("Please press one of the following keys:\n");
  printf("Y-Erase TOTAL-SUM & confirm.\n");
  printf("P-Print current value of TOTAL-SUM.\n");
  printf
     ("A-Add count value to TOTAL-SUM & print value.\n");
  printf("E-Erase TOTAL-SUM, confirm & exit.\n");
  for (;;)
     {
     key = getch();
     if (key == 'Y')
      {
      total_sum = 0;
      printf("TOTAL-SUM erased\n");
      }
     else if (key == 'P')
      printf("TOTAL-SUM is %d\n", total_sum);
     else if (key == 'A')
      {
      total_sum += count;
      printf("TOTAL-SUM is %d\n", total_sum);
      }
     else if (key == 'E')
      {
      total_sum = 0;
      printf("TOTAL-SUM erased, exiting\n");
      return;
      }
     }
}
```

לפני שנתכנת דוגמה זו בעזרת מנגנון **switch**, נציג את כללי התחביר שלו:

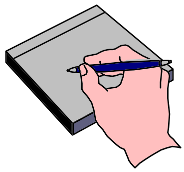
```
switch (<ביטוי>)
   {
     case <קבוע> :
          <משפט> ;
            ...
          break ;
     case <קבוע> :
          <משפט> ;
            ...
          break ;
     case     :
            ...
     default  :
          <משפט> ;
            ...
            ...
}
```

- בתוך <**ביטוי**> מוגדר משתנה הבקרה שלפיו יתבצע הפיצול.
- כל אחת ממילות **case** מלווה בערך קבוע של אותו משתנה בקרה.
- המשפט **break** משמש לסיום קטע הפיצול, ובלעדיו תתבצע גם שארית התוכנית המופיעה אחרי הקטע המתאים. רק אם אנו רוצים למזג קטעים עוקבים, נוותר על משפט זה. פעולת מיזוג קטעים עוקבים נקראת **גלישה**.
- המילה **default** וקטע התוכנית שאחריה הם בגדר אפשרות (כלומר, אינם חובה). כאשר הם כתובים, יתבצע קטע התוכנית אם ערך המשתנה אינו תואם אף לא אחת מהאפשרויות. שים לב שהמילה default אינה חייבת להופיע בסוף משפט switch, אלא יכולה להופיע בכל מקום ב"תוכו". תקן הכתיבה של משפט switch מחייב התייחסות ל**ברירת המחדל** (default).

נחזור לדוגמה הקודמת:

```
        /*****      chp0409.c :קובץ תקליטור       *****/
#include <stdio.h>
#include <conio.h>
void main(void)
{
   int count=8, total_sum=0;
   char key;
   printf("Please press one of the following keys:\n");
   printf("Y-Erase TOTAL-SUM & confirm.\n");
   printf("P-Print current value of TOTAL-SUM.\n");
   printf("A-Add count value to TOTAL-SUM & print value.\n");
   printf("E-Erase TOTAL-SUM, confirm & exit.\n");
```

```
  for (;;)
     {
     switch (key = getch())
      {
      case 'Y':
          total_sum = 0;
          printf("TOTAL-SUM erased\n");
          break;
      case 'P':
          printf("TOTAL-SUM is %d\n", total_sum);
          break;
      case 'A':
          total_sum += count;
          printf("TOTAL-SUM is %d\n", total_sum);
          break;
      case 'E':
          total_sum = 0;
          printf("TOTAL-SUM erased, exiting\n");
          return;
          default: printf("default");
      }
     }
}
```



הערה!

*ראוי להזכיר שוב את חשיבות משפט **break**, כי בלעדיו יתבצעו קטעי תוכנית שאינך מעוניין בהם: כאשר הערך של משתנה הבקרה שווה לקבוע שליד המילה case, יתבצעו **כל** המשפטים שאחריו, זה אחר זה עד סיום משפט הבקרה. אפשר להפסיק רק באמצעות הוראת break. משפט break משמש ל"יציאה" מתוך משפט בקרה כלשהו, ופעולתו כאן דומה.*

את קטע התוכנית הקודם ניתן לקצר, על ידי שימוש סלקטיבי במשפט break.

```
        /*****      chp04010.c :קובץ תקליטור       *****/
#include <stdio.h>
#include <conio.h>
void main(void)
{
  int count=8, total_sum=0;
  char key;
  printf("Please press one of the following keys:\n");
  printf("Y-Erase TOTAL-SUM & confirm.\n");
```

```
  printf("P-Print current value of TOTAL-SUM.\n");
  printf("A-Add count value to TOTAL-SUM & print value.\n");
  printf("E-Erase TOTAL-SUM, confirm & exit.\n");

  for (;;)
    {
    switch (key = getch())
     {
     case 'Y':
         total_sum = 0;
         printf("TOTAL-SUM erased\n");
         break;
     case 'A':
         total_sum += count;
     case 'P':
         printf("TOTAL-SUM is %d\n", total_sum);
         break;
     case 'E':
         total_sum = 0;
         printf("TOTAL-SUM erased, exiting\n");
         return;
     }
    }
}
```

כפי שניתן לראות, שינינו את סדר המשפטים והסרנו את אחד ממשפטי break. כך אנו מנצלים את העובדה שאותו משפט הדפסה מתבצע עבור שני ערכי משתנה הבקרה ('A' ו-'P').

הרחבה של משפט IF..ELSE

בשפת C קיים **מבנה מיוחד** המאפשר לשלב את הפונקציונליות של משפט if..else בתוך ביטוי, או כחלק ממנו. משפט זה מאפשר בחירה בין שני ביטויים על פי ערכו של תנאי מסוים. עם זאת, בניגוד למשפט if..else, ניתן לשלבו **בתוך משפט הצבה**, או בתוך ביטוי לוגי. משפט זה נקרא "**משפט הצבה מותנה**" (Conditional Assignment), או **ביטוי מותנה** (Conditional Expression).

התחביר של המשפט:

(<תנאי>) ? (<ביטוי 1>) : (<ביטוי 2>) ;

במשפט זה יבוצע <**ביטוי 1**> אם התנאי מתקיים; <**ביטוי 2**> יבוצע אם התנאי אינו מתקיים. שים לב, שבניגוד למשפט if..else, **לא ניתן** לבצע קטעי תוכנית בכל אחת מהאפשרויות, אלא **רק ביטוי בודד**.

דוגמאות:

```
result = (x != 0) ? 10000 / x : 0 ;
printf("Result = %d/n", (x != 0) ? 100 : 0) ;
```

דוגמת תכנות:

```
        /*****      chp04011.c  :קובץ תקליטור       *****/

#include <stdio.h>

void main(void)
{
  int i;
  char chr;

  i = 5;
  chr = (i >= 10) ? 'A' + i : 'a' + i ;
  printf("Character is %c \n",chr);
}
```

```
Character is f
```

דוגמאות נבחרות

א. לוח הכפל

הדוגמה הבאה מציגה את העוצמה והאפשרויות של **משפט for**. בעזרת שני משפטים ניתן להדפיס לוח-כפל מ-1 עד 10 (או טווח מספרים אחר כלשהו. שים לב לשינוי הכותרת):

```
        /*****      chp04012.c  :קובץ תקליטור       *****/

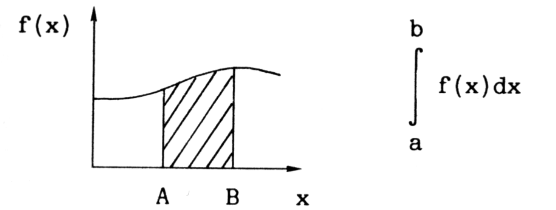
#include <stdio.h>

void main(void)
{
  int   i, j;
                    /* table header */
  printf("Multiplication-table using -for- loop \n\n");
  printf
  ("--------------------------------------------\n");
  printf
  ("XX| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |\n");
  printf
```

```
  ("---------------------------------------------\n");
                   /* program */
  for (i=1 ; i <=10 ; ++i)
   {
     printf("%2d|", i);
     for (j=1 ; j <= 10 ; ++j)
     printf("%3d|", i*j);
     printf("\n");
   }
}
```

ב. חישוב אינטגרל מסוים

ערך האינטגרל המסוים עבור פונקציה כלשהי, הוא השטח שמתחת לאותה פונקציה בין נקודות הקצה. בתרשים 4.6 - זהו השטח המקווקו.



תרשים 4.6: חישוב אינטגרל

לפניך תוכנית פשוטה שתאפשר לך למצוא ערך של כל אינטגרל בכל דרגת דיוק רצויה, על ידי שינוי הפרמטרים והפונקציה המחושבת.

בדוגמה נתייחס לפונקציה: $X^3 + X + 5$

ונחשב אינטגרל בגבולות: 0.25 עד 2.55, כאשר $dx = 0.01$

```
        /*****      chp04013.c  :קובץ תקליטור        *****/

#include <stdio.h>

void main(void)
{
  float lo_limit,hi_limit,delta_x,result,x_val;

                   /* initialization */
  x_val = lo_limit = 0.25;  /* init x value */
  hi_limit = 2.55; /* high integral value */
  delta_x = 0.01;  /* dx=0.01 */
  result = 0;
  while (x_val < hi_limit)
```

```
     {
     result += x_val*x_val*(x_val+5)*delta_x;

     x_val += delta_x;
     }

  printf("Result of integral is %f\n", result);
}
```

כדי לחשב אינטגרל באזור אחר, עליך לשנות את הגבולות hi_limit, lo_limit, וכדי לשנות את דרגת הדיוק עליך לשנות את delta_x.

תרגילים

תשובות בתקליטור: c:\tclite\targilim\c4e1-c4e17

1. מה יהיו ערכי המשתנים x ו-r בסיום התוכנית הבאה:

```
main()
{
int r, x;
r = 20
do
   {
   x = r--;
   if (x <6)
      r = 30;
   }
while ((r !=30) && (--x >5));
}
```

2. מה יהיו ערכי המשתנים m ו-p בסיום התוכנית הבאה:

```
main()
{
int m;
float p;

for (p = 1.05, m = 30; m; --m)
   p *= 2;
printf("The value of p is %f", p-20);
}
```

3. מה יהיה הפלט של התוכנית הבאה?

```
main()
{
int i, j;

for (i = 0; i< 10; ++i)
   for (j = i; j < 10; ++j)
      printf("%d", j * ++i);
}
```

4. מה יהיה הפלט של התוכנית הבאה, עבור הקלט: A, 3, M, C, E?

```
main()
{
  char select = ' ';
  int num = 1 ;

   while (select != 'q')
      switch (select = getch())
         {
          case 'E' : if ( num > 5 )
                        select = 'q' ;
                     break ;

          case 'M' : num = 6 ;
                     printf ("%d" , num ) ;

          case 'C' : num += select ;
                     printf ("%d" , num ) ;
                     break ;
         }
}
```

5. האם התוכנית הבאה תסתיים? הסבר.

```
main()
{
int i, num;
char select;
float flt;

flt = 0.5;
num = -6 ;
for (i=5; i > -5; --i)
   {
   num += 1;
   while (i++ > 0)
      i += num * flt
      }
}
```

6. כתוב תוכנית הקולטת מספר בן 2 ספרות (בעזרת getch()) ומדפיסה את העצרת שלו (n!).

7. כתוב תוכנית המדפיסה את שבעת האיברים הראשונים בסדרת פיבונצ׳י המתחילה במספר 3.

זכור, כי בסדרת פיבונצ׳י: $a_{n+1} = a_n + a_{n-1}$.

לדוגמה: `1, 2, 3, 5, 8, ...`

8. כתוב תוכנית הבודקת אם מספר כלשהו בעל שתי ספרות הוא ראשוני. הבדיקה תיעשה על ידי חלוקת המספר n בכל המספרים שקטנים ממנו, כדי לראות אם כל החלוקות הסתיימו בשארית. הדפס הודעות מתאימות.

9. הדפס טבלה שתכיל עבור כל מספר מ-0 עד 10 את החזקות הריבועיות והמשולשות שלו. עליך לערוך את הפלט כפי שמוצג בטבלה הבאה:

number	square	triple
0 1	4 תווים	5 תווים

10. כתוב תוכנית המוצאת את השלם הגדול ביותר שריבועו קטן מ-35678.

התייחס למגבלות הטיפוס int. כתוב תוכנית יעילה בזמן ובכמות המשתנים.

הדפס את תוצאת התוכנית בליווי הודעה מתאימה.

11. כתוב פקודת if פשוטה, בעלת תנאי יחיד, שתהיה שקולה לפקודה המוצגת כאן:

```
if (x < 0)
   {
   if (y < 2)
      m = x + y;
   }
else
   if (x == 1)
      m = x + y;
```

12. בהתייחס לשאלה 10, כתוב תוכנית המחשבת פתרון מקורב של משוואה ריבועית $Ax^2 + Bx + C = 0$, אשר מקדמיה הם A, B ו-C. הדפס את השורשים שקיבלת, בהנחה שהפתרונות ממשיים.

13. כתוב תוכנית המדפיסה 4 מספרים שלמים בסדר יורד.

14. כתוב תוכנית הקולטת תו מהמקלדת ומסווגת אותו לאחת הקבוצות הבאות:

א. אותיות גדולות

ב. אותיות קטנות

ג. ספרות

ד. סימנים אחרים

הדפס עבור כל תו קלט את הסיווג שלו.

15. מה יתבצע בתוכנית הבאה?

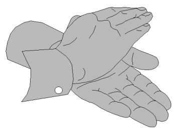
```
main()
{
int i, j, k;
j = 10; k = 2;
for (i = 0; i < 100; ++i)
   {
   j *= (i + 1);
   k = j * i;
   if (i > 5)
      continue;
   if (i > 20)
      break;
   }
}
```

16. כיצד תשכתב את התוכנית שבשאלה 15 כדי להימנע מהפקודות continue ו-break?

17. קיימים 4 מספרים קטנים מ-500 בעלי תכונה מיוחדת. אם לוקחים כל ספרה במספר, מעלים אותה בחזקת שלוש ומחברים את התוצאות, הסכום יהיה שווה למספר המקורי.

לדוגמה: $1^3 + 5^3 + 3^3 = 153$

מצא לפחות עוד מספר אחד כזה. עשה זאת באמצעות תוכנית מחשב מתאימה.



משימות:

תרגילים 1-3.

פרק 5

פונקציות

שפת C מאפשרת הגדרה נוחה של **פונקציות**. השימוש בפונקציות מאפשר חלוקה של משימת תכנות גדולה למספר משימות קטנות ועצמאיות. בכך משיגים שתהליך הפיתוח והכתיבה של תוכנית מחשב גדולה יהיה פשוט יותר והדרגתי. כל פונקציה מתכנתים ובודקים כיחידה עצמאית, ורק לאחר שמשוכנעים שהיא פועלת כראוי, משלבים אותה בתוכנית הכללית.

אם לדוגמה, יש לכתוב תוכנית המבצעת הדמייה למחשבון כיס, ואחת הפעולות המבוצעות במחשבון היא העלאה בריבוע, נכתוב, למשל, פונקציה בשם square(), שתקבל נתון אחד ותחזיר את תוצאת הריבוע שלו. את הפונקציה הזו ניתן לבדוק בנפרד, כדי לוודא שאכן היא מפיקה את התוצאה הנכונה, מבלי שיהיה צורך לבדוק את התוכנית כולה. רק אחר כך נכלול אותה יחד עם פונקציות אחרות שבדקנו קודם, כדי להרכיב יחד את התוכנית הכוללת.

פיתוח הפונקציות יכול להיעשות על ידי מתכנת בודד, או על ידי מספר מתכנתים. ניתן לכלול בתוך תוכנית חדשה פונקציות שנכתבו ונבדקו עבור תוכניות קודמות וביצעו תפקידים מוגדרים החוזרים על עצמם בתוכניות שונות (כמו לדוגמה חישוב סכום של סדרת מספרים, או הדפסת כותרת של דוח), וכך נחסוך זמן וטעויות, והעיקר - **לא** "נמציא את הגלגל מחדש". פונקציות קלט/פלט של ספריית המערכת מהוות דוגמה לאפשרות זו, כי הן נכתבו מראש על ידי מתכנת אחר, וניתנות לשימוש הכל.

ניתן לפנות לפונקציה מסוימת ממקומות שונים בתוכנית ולהימנע מן הצורך לשלב את קטע התכנות המגדיר אותה בכל אותם מקומות, וכך ניתן לחסוך ב"נפח" התוכנית. פעמים רבות התהליך הוא הפוך: במהלך הכתיבה של התוכנית קורה, שקטע תכנות מסוים חוזר על עצמו פעמים אחדות. במרבית הפעמים מומלץ להפוך קטע זה לפונקציה, ולפנות אליה מכל המקומות בתוכנית, כפי הצורך.

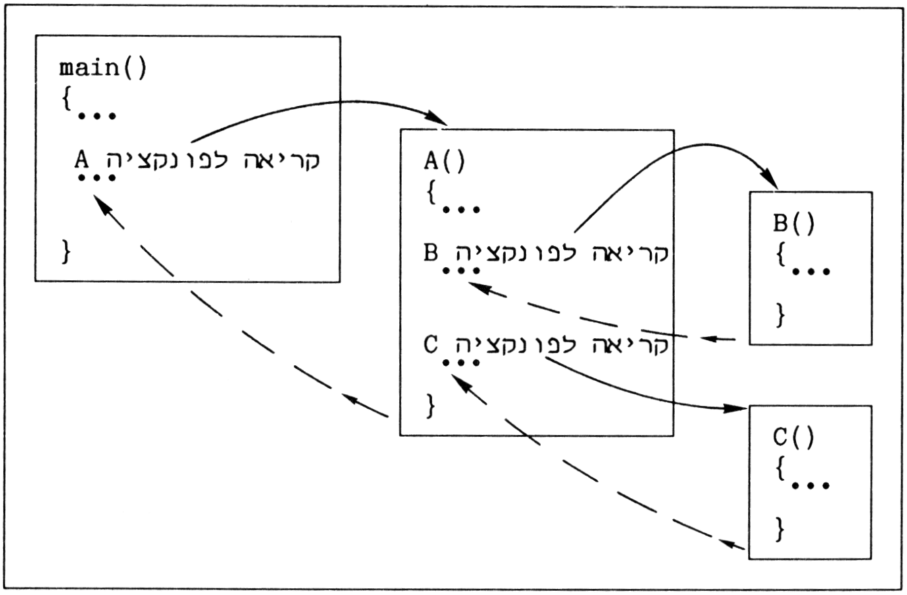
המשתנים המוגדרים בתוך הפונקציה הם **משתנים פרטיים** שלה, אשר מוכרים אך ורק בתוכה. לפיכך, יכולים להיות בתוכנית פונקציות אחרות, שקיימים בהן משתנים בעלי שמות זהים. בהמשך נלמד על דרכים לעקוף הפרדה זו, כדי לשתף משתנים בין הפונקציות. נציין רק, ש**שיתוף משתנים** חייב להיות יזום במפורש על ידי המתכנת.

עצמאות הפונקציות והפרדת המשתנים הן שתי תכונות חשובות לקיום גמישות השימוש במנגנון הפונקציות, והיכולת להרכיב תוכנית מקטעים שנכתבו על ידי מתכנתים שונים, מבלי שייגרמו שיבושים כתוצאה מכך.

הקישור בין הפונקציה לתוכנית הקוראת לה מתבצע באמצעות סדרת ערכים הנקראים בשם הכללי - **ארגומנטים, או פרמטרים פורמליים**. **הערכים המועברים מן התוכנית הקוראת אל הפונקציה** בשלב הקריאה לה קרויים **פרמטרים אקטואליים** (Actual Parameters). הפונקציה מתבצעת ומחזירה בסיומה **ערך בודד** הקרוי **ערך מוחזר** (Returned Value), אשר מהווה את **תוצאת הביצוע**. לנושא זה נחזור בהמשך.

הפנייה אל הפונקציה גורמת להפסקה זמנית במהלך ביצוע התוכנית הקוראת; הבקרה מועברת לפונקציה; ורק לאחר שהפונקציה מסתיימת מוחזרת הבקרה לתוכנית הקוראת, אשר ממשיכה את הביצוע מנקודת ההפסקה. פונקציה אחת יכולה לקרוא לפונקציה נוספת, וכך ליצור שרשרת של קריאות לפונקציות. **בכל קריאה כזו מועברת הבקרה לפונקציה הנקראת ומוחזרת בסיומה לפונקציה הקוראת.**

בתרשים 5.1 ניתן תיאור סכימטי של קריאה לפונקציות בשתי רמות עומק:



תרשים 5.1: מהלך הקריאה לפונקציה - קינון קריאות

הגדרה של פונקציה

כל פונקציה מזוהה בתוכנית **בשם ייחודי**. אורך השם והחוקים החלים עליו **זהים** לחוקים החלים על **שמות משתנים**. הצורה הכללית של הגדרת פונקציה:

```
<function_name> (טיפוס<arg1>,טיפוס<arg2>...טיפוס<argn>)
   {  ...
      <גוף הפונקציה>
      ...
   }
```

<function name> שם הפונקציה.

<arg$_1$>...<arg$_n$> רשימה של שמות משתנים המהווים ערכים (פרמטרים) לפונקציה. ערכים אלה קרויים **פרמטרים פורמליים** (Formal Parameters), אשר נשמרים בזיכרון כמשתנים זמניים.

הזכרנו קודם, שהתוכנית הקוראת מספקת לפונקציה סדרת ערכים, אשר נקראים **"פרמטרים אקטואליים"**. סדרת הערכים האלה **מועתקת** לתוך הפרמטרים הפורמליים (המשתנים arg$_1$ ...arg$_n$) על פי סדר הופעתם. המשתנים הפורמליים (שתפקידם "ייצוגי" בלבד) **מקבלים** את ערך הפרמטרים האקטואליים בזמן ריצת התוכנית, ואלה הם **הערכים התחלתיים** שלהם בעת הקריאה הנוכחית לפונקציה.

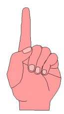
הערכים שהתוכנית הקוראת מעבירה לפונקציה יכולים להיות קבועים, משתנים או ביטויים. לאחר העתקת ערכם למשתנים הפורמליים **ניתק הקשר ביניהם**, ועיקרון העצמאות של הפונקציה נשמר. שיטת קישור זו, בין התוכנית הקוראת לבין הפונקציה, הינה **"העברה לפי ערך"** (Call by Value), אשר הינה התקן בשפת C. היא נקראת כך, מכיון **שמועברים רק הערכים של הפרמטרים האקטואליים - ולא הכתובת שלהם.**

הפרמטרים הפורמליים יכולים להיות מטיפוסים שונים, והגדרתם נחשבת חלק מהגדרת הפונקציה. בקריאה לפונקציה עליך להקפיד על מספר פרטים חשובים:

א. **מספר** הפרמטרים האקטואליים צריך להיות זהה למספר הפרמטרים הפורמליים.

ב. אם **טיפוס** פרמטר פורמלי אינו זהה לטיפוס של הפרמטר האקטואלי המתאים, יבצע מהדר C המרה מטיפוס לטיפוס, על פי החוקים הנהוגים במשפט ההצבה.

ג. סדר הפרמטרים בקריאה לפונקציה צריך להיות זהה לסדר שהוגדר בה.



שים לב!

מרבית המהדרים הקיימים אינם בודקים אחד או יותר מהפרטים האלה, וטעות בקריאה לפונקציה תתגלה רק בזמן ביצוע התוכנית.

ד. כאשר פונקציה אינה מקבלת ערך כלשהו, משתמשים בטיפוס הריק (void) כדי לציין עובדה זו. לדוגמה:

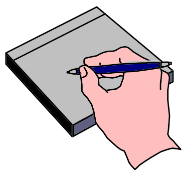
```
int main (void)
{
}
```

ערך מוחזר

בסיום ביצוע הפונקציה ניתן להעביר לתוכנית הקוראת **ערך בודד**, המהווה את תוצאת הביצוע של הפונקציה. לערך זה קוראים **ערך מוחזר** (Returned Value). המשפט "**return(<ערך>)**" נועד להחזיר ערך מהפונקציה, כאשר <ערך> יכול להיות קבוע, משתנה, או ביטוי.

המשפט **return** גורם גם לסיום מהלך ביצוע הפונקציה וחזרה לתוכנית הקוראת, עם הערך המוחזר. משפט return **אינו** חייב להיות המשפט האחרון בפונקציה, ואינו חייב להיות היחיד. ניתן ל"צאת" מהפונקציה ממספר מקומות (למשל, במשפט switch), אך רצוי שתהיה יציאה אחת בלבד.

משפט return "ריק" (;return), ללא פרמטר כלשהו, יגרום לחזרה לתוכנית הקוראת **ללא ערך מוחזר**. אפשר גם לכתוב פונקציות ללא משפט return. במקרה זה תתבצע **חזרה בלתי מפורשת,** לאחר ביצוע **הפקודה האחרונה** בפונקציה. כאשר המהדר מגיע לסיום הפונקציה, הוא יכניס משפט חזרה בתוכנית המתורגמת שהוא מפיק, למרות שמשפט זה אינו נמצא במקור.



הערה!

קיימים מקרים שבהם מציינים פרמטר של ערך מוחזר מפונקציה, בשעה שהיא אינה מחזירה ערך כזה. המהדר אינו מודיע על טעות, אך בזמן ריצת התוכנית יתכנו שיבושים, שכן הערך המוחזר אינו מוגדר. במהדרים מודרניים הפועלים לפי תקן ANSI, יש על כך הודעת מהדר!

דוגמאות לפונקציות

נציג מספר דוגמאות לתכנות פונקציות באמצעות שעשוע ב"מחשבון".

נניח שקנינו מחשב, ש "יודע" לחבר זוגות של מספרים בלבד, ולחשב את הערך השלילי של מספר נתון. באמצעות מנגנון הפונקציות של שפת C נוכל לשפר את יכולת המחשב ולאפשר לו לבצע פעולות מתמטיות נוספות: כפל, חילוק וחזקות. לתכנות הפעולות החסרות במחשב נגדיר פונקציות, פרמטרים פורמליים וערכים מוחזרים כנדרש.

פונקציית כפל

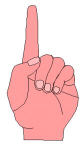
שם:	mult	
פרמטרים פורמליים:	md	נכפל
	mr	כופל
ערך מוחזר:	md * mr	זכור שפעולת הכפל לא קיימת במחשבון שלנו

פונקציית חילוק

שם:	divd	
פרמטרים פורמליים:	dd	מחולק
	dr	מחלק
ערך מוחזר:	r	הערך **השלם** של החילוק dd / dr

פונקציית שארית

שם:	rmnd	
פרמטרים פורמליים:	dd	מחולק
	dr	מחלק
ערך מוחזר:	r	**השארית** מפעולת החילוק dd / dr



שים לב לנקודות הבאות!

א. תכנון הפונקציות נעשה ללא פירוט אופן התכנות שלהן. ההגדרות כוללות רק את "הקליפה החיצונית", הכוללת: שם פונקציה, פרמטרים וערך מוחזר.

ב. כל פונקציה הינה עצמאית וניתנת לתכנות בכל דרך רצויה. התנאי המחייב היחיד הוא החזרת ערך.

*ג. למרות זהות השמות של הפרמטרים הפורמליים בפונקציות divd() ו-rmnd(), המשתנים בכל אחת מהן **אינם** קשורים ביניהם.*

דרך תכנון זו דומה מאוד לתכנון של פרויקטים גדולים. באמצעותה ניתן לפשט את תהליך הפיתוח, לקבל "תמונה" כוללת, ורק אחר כך להתרכז בביצוע בפועל.

פונקציית כפל - mult()

```
     /*****        chp0501.c :מתוך קובץ תקליטור        *****/

(1)    mult(int md, int mr)
(2)    {
(3)      int   res;

(4)      for (res=0 ; mr ; mr--)
(5)        res = res + md;
(6)      return(res);
(7)    }
```

הסבר:

א. שורה 1: הגדרה של שם הפונקציה, רשימת הפרמטרים הפורמליים שלה וטיפוסיהם. md ו-mr הם פרמטרים פורמליים מטיפוס שלם.

ב. שורה 3: הגדרה של משתנה עזר res, לצורך חישוב התוצאה. זהו "משתנה פנימי" של הפונקציה, אשר מוכר אך ורק בתוכה. אם קיים משתנה בעל שם זהה בפונקציה אחרת - **לא** יהיה ביניהם כל קשר.

ג. שורות 4-5: חישוב של תוצאת הכפל באמצעות לולאת for, שבתוכה מבוצעת פעולת חיבור. בחרנו בלולאה זו כדי לאתחל את המשתנה res בערך 0, ולעדכן את מונה הלולאה, שהוא גם תנאי העצירה שלה.

הלולאה מתבצעת כל עוד mr (הכופל) שונה מ-0 (בפרקים הקודמים הסברנו מדוע X וגם X != 0 הם תנאים זהים בשפת C). בכל מהלך של הלולאה מופחת 1 ממונה הלולאה mr ותוצאת החיבור מוגדלת ב-md (הנכפל).

ד. שורה 6: בסיום הביצוע של הלולאה, כאשר mr=0, המשתנה res מכיל את התוצאה. ערך זה מוחזר לתוכנית הקוראת באמצעות הפקודה return.

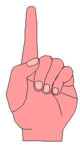
ה. תוצאת הפונקציה תהיה נכונה לכל ערך התחלתי של mr, אשר גדול או שווה אפס. ערך שלילי יגרום לתוצאה שגויה (בהמשך נראה כיצד אפשר לפתור בעיה זו).

לפניך **פונקציה המבצעת פעולה זהה**, אשר תוכנתה בדרך שונה, ומנצלת אפשרויות נוספות של השפה:

```
mult(int md,int mr)
{
  int res ;

  for (res=0 ; mr-- ;)
    res += md ;

  return (res) ;
}
```



שים לב לשינויים בלולאה!

א. בדיקת תנאי הסיום שולבה בהקטנת מונה הלולאה.

ב. הגדלת res בערך הנכפל נעשית על ידי משפט הצבה בכתיבה מקוצרת.

פונקציית חילוק - divd()

```
     /*****       chp0501.c :מתוך קובץ תקליטור       *****/

(1)    divd(int dd, int dr)
(2)    {
(3)      int   try;
```

```
(4)       for (try=1 ; try <= (dd+1) ; ++try)
(5)         if (mult(dr, try) > dd)
(6)             return(try-1);
(7)     }
```

הסבר:

א. החלוקה מתבצעת על ידי "ניחוש" התוצאה ובדיקה בפעולת כפל. כדי לנסות את כל הניחושים האפשריים, בנינו לולאה המגדילה בכל פעם את ערך הניחוש ב-1.

ב. בפונקציה זו ניצלנו את פונקציית הכפל המחזירה תוצאות כפל של שני מספרים. משפט if בשורה 5 בודק את הערך המוחזר כנגד התוצאה הרצויה.

ג. אם משפט התנאי מתקיים, הרי שהצלחנו ל"נחש" את המנה (לאחר הפחתת אחד, עקב אופי התנאי). נזכיר רק, שהערך המוחזר הוא הערך השלם של החלוקה.

ד. שים לב לשימוש בפונקציה ()mult (שכבר הגדרנו) בגוף הפונקציה החדשה ()divd.

פונקציית שארית - ()rmnd

```
    /*****      chp0501.c  :מתוך קובץ תקליטור      *****/

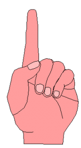
(1)  rmnd(int dd,int dr)
(2)  {
(3)    return(dd - mult(divd(dd,dr), dr));
(4)  }
```

הסבר:

א. פונקציית השארית מדגימה היטב את אחד המאפיינים החשובים בשפת C. הפונקציה (ולמעשה, הערך המוחזר על ידה) **זהה** לביטוי, משתנה או קבוע. כתוצאה מכך, ניתן להשתמש בה בכל מקום בו מותר להשתמש בביטוי או במשתנה, כפי שמוצג בשורה 3.

לדוגמה, הפרמטר האקטואלי הנמסר לפונקציה ()mult כנכפל, הוא למעשה התוצאה המוחזרת של הפונקציה ()divd. מנגנון זה מאפשר **להעביר ערכים בין פונקציות ללא צורך בשמירתם במשתנים**.

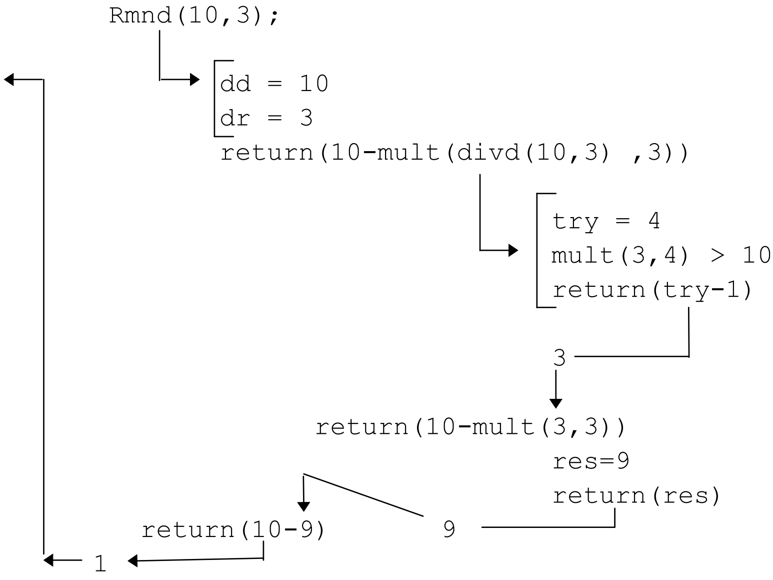
ב. מכיון שהפונקציה ()divd מחזירה את המנה השלמה של פעולת החילוק (dd/dr), הרי שמכפלתה ב-dr תיתן תוצאה קטנה, או שווה ל-dd. ההפרש בין המספר המקורי והמכפלה מהווה את שארית החילוק, והוא מוחזר כערך בפקודה return. גם פעולת החיסור מדגימה היטב את היכולת לחסר ערך מוחזר של פונקציה, כאילו היה ביטוי, או משתנה.



שים לב!

*הפונקציה (ולמעשה, הערך המוחזר על ידה) **זהה** לביטוי, משתנה או קבוע. כתוצאה מכך, ניתן להשתמש בפונקציה בכל מקום בו מותר להשתמש בביטוי או במשתנה.*

בתרשים 5.2 ניתן תיאור סכימטי של מהלך הביצוע לאחר הקריאה לפונקציה ()rmnd, כאשר המחולק שווה 10 והמחלק שווה 3.



תרשים 5.2: *מהלך ביצוע של פונקציה*

עתה בידינו מספר פונקציות ייעודיות כדי לבצע פעולות חישוב במספרים שלמים, ללא שימוש בפקודות כפל וחילוק של המחשב. בתוכנית הבאה נדגים מספר שימושים בפונקציות אלו.

```
      /*****      chp0501.c :מתוך קובץ תקליטור        *****/

#include <stdio.h>

void main(void)
{
   int x, y, z, r, i;

   x = 3;
   z = 20;

   y = mult(x, z);
   printf("%d * %d = %d\n", x, z, y);

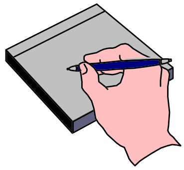
   i = divd(z,x);
   r = rmnd(z,x);
   printf("%d / %d = %d %d/%d\n", z, x, i, r, x);
}
```

לאחר הרצת התוכנית נקבל את הפלט הבא:

```
3 * 20 = 60
20 / 3 = 6 2 / 3
```

חזרה לפונקציית הכפל

נחזור לבעיה שהתעוררה בפונקציית הכפל, כאשר ערכו של הכופל mr בזמן הקריאה, הוא שלילי. כזכור, סיום הלולאה בפונקציה ()mult מותנה בקיום התנאי --mr השקול לתנאי mr != 0. נשתדל לפתור את הבעיה מבלי לפגוע בהגדרה הראשונית של הפונקציה, וכדי שלא נצטרך לשנות את הפונקציות בהן קראנו לפונקציה ()mult. דרישה זו חשובה ביותר כאשר עוסקים בתוכניות גדולות ומורכבות. אי עמידה בדרישה זו עלולה לגרום לשינויים גדולים והפסדי זמן משמעותיים. לפנינו רק דוגמה, אולם במציאות אנו עומדים מדי פעם בפני הצורך לתקן ולעדכן פונקציות שונות שנכתבו.



הערה חשובה!

*הדרישה לשמור על **עקביות ואחידות של תנאי ה"כניסה" וה"יציאה"** הינה בעלת משמעות חשובה.*

נעתיק את הפונקציה ()mult לפונקציית העזר ()mul. היא תשרת את הפונקציה ()mult, כאשר mr גדול או שווה ל-0 בלבד. הבדיקה שאכן mr>=0 תהיה באחריות הפונקציה החדשה שנכתוב.

לפניך שתי הפונקציות המקיימות את הדרישה לפעול הן במספרים חיוביים והן במספרים שליליים.

```
/*****      chp0502.c :מתוך קובץ תקליטור      *****/

     mul(int mdd, int mrr)
         {
         int   res;

         for (res=0 ; mrr-- ; )
          res += mdd ;

         return (res) ;
     }

(1)  mult(int md, int mr)
(2)  {
(3)    if (mr >=0)
(4)      return (mul (md, mr));
(5)    else
(6)      return (-mul (md, -mr)) ;
(7)  }
```

הסבר:

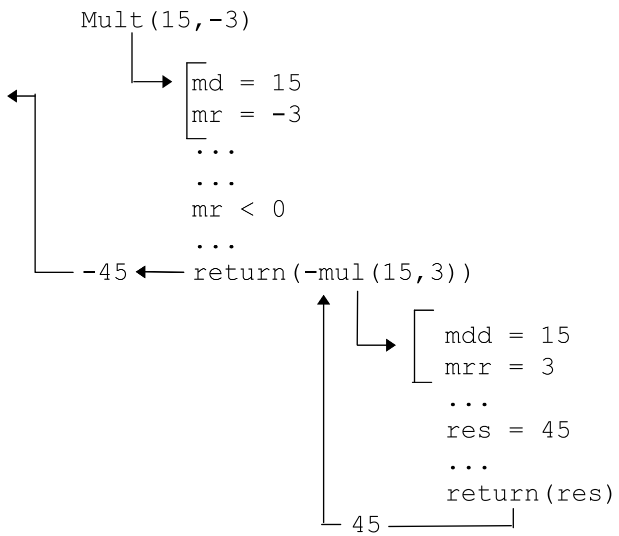
א. הפונקציה ()mult החדשה בודקת בשורה 3 את ערך הכופל mr. אם הוא חיובי, מופעלת הפונקציה ()mul (שהיא פונקציה ()mult הקודמת) והתוצאה שלה מוחזרת על ידי ()mul. אם הכופל הוא שלילי, מתבצעת שורה 6 והפונקציה ()mul מופעלת כאשר mr מופיע עם סימן שלילי (ההופך את ערכו לחיובי). תוצאת ()mul מוחזרת בסימן שלילי, ועל ידי כך קיבלנו את התוצאה הרצויה.

ב. שים לב, שבפונקציה החדשה ()mult אין צורך במשתנה העזר res, מכיון שמשימת החישוב מוטלת כעת על הפונקציה ()mul. הפונקציה ()mult משמשת כמתווכת בין הפונקציה שקראה לה (ובה רוצים לכפול מספרים חיוביים ושליליים), לבין הפונקציה ()mul המחייבת קבלת כופל חיובי.

ג. הפונקציות הקוראות ל-()mult **אינן "יודעות"** על השינוי שחל בה. עבורן, הפונקציה ממשיכה לקבל שני ערכים (כופל ונכפל) ומחזירה את המכפלה שלהם. ההבדל הוא כמובן, בהסרת המגבלה של מספרים חיוביים, אשר הייתה במקרה הראשון.

ד. שים לב לעובדה, ש**אין קשר** בין הכופל mr בפונקציה ()mult לבין הכופל mr בפונקציה ()mul. כפי שציינו קודם, בשפת C מועברים הפרמטרים "לפי ערך". כאשר ()mul מופעלת עם הפרמטר (mr-), הפרמטר הפורמלי שלה mrr מקבל את הערך הדרוש ובזה **מסתיים הקשר** בין שתי הפונקציות. הקשר מתחדש כדי לקבל את הערך המוחזר. mrr ו-mdd הם משתנים שונים ועצמאיים, אשר מוכרים רק בפונקציה שבה הוצהרו.

בתרשים 5.3 מוצגת דוגמת מכפלה של מספר שלילי.



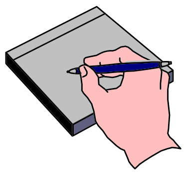
תרשים 5.3: מכפלה של מספר שלילי

ערכים מוחזרים וטיפוסיהם

עד עתה עסקנו בפונקציות המחזירות ערכים מטיפוס שלם (int), ועל כן הן נקראות **פונקציות מטיפוס שלם**. כאשר מגדירים פונקציה חדשה ולא מציינים דבר (כלומר, איזה ערך יוחזר), היא תוכר כפונקציה מטיפוס שלם, שכן זוהי **ברירת המחדל** (Default Option) בשפת C. אפשר להגדיר פונקציה מטיפוס אחר, לדוגמה שהיא תחזיר ערך מסוג float, char או double, אך לשם כך יש להשתמש ב**משפט הגדרה,** כמפורט להלן:

<טיפוס הפונקציה> <שם הפונקציה> **(** <הגדרת טיפוסי הערכים>
<רשימת ערכים פורמליים>**)**

{
<הגדרת משתנים פנימיים>
...
<גוף הפונקציה>
...
}



הערה!

הטיפוס של הפונקציה הינו למעשה הטיפוס של הערך המוחזר שלה. יש בכך היגיון רב, מכיון שערך זה ישובץ אחר כך במקום שבו הופיעה הקריאה לפונקציה.

כאשר פונקציה או תוכנית קוראים לפונקציה מטיפוס שלם, עליהם **להכריז** על טיפוס הפונקציה. למרות שהטיפוס השלם הוא ברירת מחדל עבור פונקציות, יש צורך להכריז על פונקציות כאלו.

ההכרזה על שימוש בפונקציה נראית כך:

<טיפוס הפונקציה> <שם הפונקציה> **( ) ;**

צמד הסוגריים הריקים מימין מציינים למהדר שזוהי הכרזה על טיפוס של פונקציה, ולא הכרזה על משתנה נוסף.

דוגמה

נרחיב את היכולת של "המחשב המחבר" שבנינו קודם, ונאפשר לו לכפול מספרים ממשיים (float) במספרים שלמים (int). לשם כך נגדיר את הפונקציה הבאה:

```
     /*****       chp0503.c  :מתוך קובץ תקליטור        *****/

     #include <stdio.h>

(1)  float fmult(float fmd, int mr)
(2)  {
(3)    float res;
```

```
(4)      for (res=0.0 ; mr-- ; )
(5)         res += fmd;

(6)   return(res);
(7)   }
```

הסבר:

א. הפונקציה ()fmult מוגדרת בשורה 1 כטיפוס float. בשורה 6 מוחזר הערך res, שהוא משתנה מטיפוס float.

ב. נזכיר שוב, שלמרות זהות השמות mr של הכופלים בפונקציות ()mult ו-()fmult, **אין כל קשר ביניהם, וכל אחד מהם מוכר בפונקציה בה הוגדר בלבד**.

ג. המגבלה המוכרת על ערך הכופל (mr >= 0) קיימת גם כאן, ועלינו לנקוט בצעדים דומים לאלה שהשתמשנו בהם קודם עבור הפונקציה ()mult.

ד. כדי להשתמש בפונקציה ()fmult יש להכריז עליה בהתאם בתוכנית הקוראת.

לפניך דוגמה של תוכנית הקוראת ל-()fmult:

```
      /*****      chp0503.c  :מתוך קובץ תקליטור       *****/

void main(void)
{
   int r;
   float f, pie;

   r = 3;
   pie = 3.1415927;
   f = fmult(pie, r);
   f = fmult(f, 2);

   printf
("Circumference of a circle of radius %d is %f\n", r, f);
}
```

לאחר הרצת התוכנית יתקבל הפלט הבא:

```
The circumference  of a circle with radius 3 is 18.849556
```

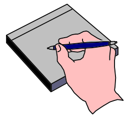
בדוגמאות הקודמות השתמשנו במילים "**הגדרה**" ו"**הכרזה**" להגדרת טיפוס של פונקציה ולהכרזת טיפוס של פונקציה. **יש הבדל משמעותי בין המילים האלו** והוא קיים גם לגבי משתנים, כפי שנפרט בהמשך.

הגדרת טיפוס הפונקציה

הגדרת טיפוס הפונקציה היא חלק מהגדרות הפונקציה כולה, ומשמשת בתהליך יצירתה של פונקציה חדשה.

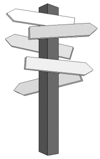
ההכרזה על טיפוס הפונקציה

ההכרזה "מודיעה" למהדר, שפונקציה מסוימת מחזירה ערך מטיפוס כלשהו. בהתאם להכרזה זו המהדר יודע כיצד להתייחס לקריאה לפונקציה זו ולערכים המוחזרים ממנה.



הערה!

אם לא מגדירים טיפוס לפונקציה, ברירת המחדל של C היא הטיפוס int.



נקודת ציון!

*הצורך במנגנון ההכרזה נובע מ**עצמאותן** של הפונקציות. למעשה, אין צורך שפונקציה הקוראת לפונקציה כלשהי תימצא באותו קובץ מקור. הקשר היחיד בין הפונקציות מתבצע באמצעות מנגנון ההכרזות.*

יש פונקציות שאינן מחזירות ערך. מטרת פונקציות אלו היא לבצע פעילות מסוימת ואין משמעות לערך מוחזר. הטיפוס של פונקציות אלו מוגדר לכן כ-void. כלומר, ללא ערך מוחזר. דוגמה:

```
void do_something(int a, int b)
{
}
```

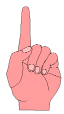
פונקציות ספריה

מהדר ++Borland C/C כמו גם מהדרי C אחרים, מספק מערכת פונקציות ספריה חזקה במיוחד אשר השקעה בלימוד שלהן כדאית. אין צורך להמציא את הגלגל מחדש, וכך ניתן לחסוך זמן יקר.

בסביבת העבודה Borland 3.1 יש 38 ספריות ובהן כ-800 פונקציות העוסקות בנושאים שונים ומגוונים כגון טיפול בקלט/פלט, מחרוזות, מסך, קבצים ועוד. אשר עוזרים מאוד במלאכת התכנות. בגרסאות מתקדמות יותר - נוספו פונקציות בנושאים שונים.

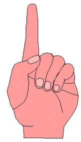
יתרון בולט בשימוש בפונקציות הוא אפשרות **השימוש החוזר**. ניתן להשתמש בפונקציה שוב ושוב על פי הצורך, הן גם אמינות למדי, כי נבדקו והופעלו על ידי מתכנתים רבים טרם שהוכנסו לספריה.

הפונקציות השונות מרוכזות על פי תפקידיהם השונים, ניתן לראות ב**קובצי הכותר** - Header files (כל הקבצים המסתיימים ב-**h.***) את הכרזות הפונקציות השונות, הגדרות טיפוס הפונקציה, פרמטרים נדרשים, הגדרות מקרו, קבועים שונים ועוד.



שים לב!

ניתן לראות וללמוד את קוד המקור של הפונקציות שכתוב ב- C, ולעיתים בשילוב עם Assembly, כפי שנמצא במערכת הפיתוח.



שים לב!

כדי להשתמש בפונקציות מספריה מסוימת, צריך לכלול את הספריה הזו בתוכנית. נעשה זאת בעזרת הפקודה הבאה, שנכתוב בראש התוכנית (עשויות להיות מספר פקודות כאלה):

```
#include <שם הספריה.h>
```

לדוגמה:

```
#include <stdio.h>
#include <alloc.h>
```

תרגילים

תשובות בתקליטור: קבצים C:\tclite\targilim\c5e1-c5e9

1. מהו פלט ההדפסה של התוכנית הבאה:

```
main()
{
int x, y, i;

x = 251;
y = 113;
for (i = 0; i < 3; ++i)
   {
      x = func1 (x, y);
      printf("Values of x and y : %d, %d /n", x, y);
   }
}

func1 (x, y)
int x, y;
{
   x *= 2;
   y += x;
   printf("func1- ->x,y :%d, %d /n", x, y);
   return (x*y);
}
```

2. הכן טבלה המתארת את טווחי ההכרה של המשתנים בתוכנית הבאה (אין תשובה בתקליטור).

```
int x1, x2;
float f1
    /* the program */
void main(void)
{
    static char c1;
    char c2;
    int i, j;
    . . .
}
long int l1;
func1 (x,y)
int x;
char y;
{
    . . .
}
func2()
{
    int x;
    float f;
    . . .
}
char r = 'A';
```

3. מה הוא פלט ההדפסה של התוכנית הבאה:

```
float f;
int func2(float f1)
{
 return ((f1<0.5)? 0:1 ) ;
}

float func1 (float flt)
{
 int i ;
 i = 1 ;
 while (i)
   {
    i = func2(flt) ;
    flt = sqrt(flt) ;
   }
 return (flt) ;
}
```

```
main()
{
 int i ;
 float flt ;
 for ( i = 12 , flt = 11.15 ; i < 14 ; ++i )
    printf ("- ->%f \n", func1 ( flt * 1.5 )) ;
 printf ("f-%f \n", f) ;
}
```

4. כתוב תוכנית המדפיסה פירמידת מספרים עבור ספרות נתונות. לדוגמה, פירמידת המספרים עבור הספרה 5 היא:

```
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
```

יש להגדיר פונקציה ()print_line, המקבלת 2 ערכים: מספר השורה (השורה העליונה היא מספר 1) וערך נוסף, שיוגדר על ידך. הפונקציה תדפיס שורת מספרים בסדר עולה ויורד, כך שהמספר המרכזי יהיה מספר השורה.

עליך לדאוג גם לסידור השורות זו מעל זו. שלב בתוכנית את הפונקציה ()get_int הקולטת ערך שלם.

5. שנה את התוכנית שבשאלה 4, כך שכאשר תבקש להדפיס מספר הגדול מ-9, היא תפסיק, ולא תדפיס את השורה כולה.

כתוב את הפונקציה ()get_int.

6. נוסחת הקירוב לחישוב e^x היא:

$$e^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+ \ ...$$

כתוב תוכנית המחשבת את הקירוב ל-e^x עבור זוג מספרים ממשיים x, eps. הקירוב יחושב בעזרת הטור האלגברי המוצג למעלה. האיבר האחרון שיחושב בסכום יקיים את היחס:

$$\frac{x^n}{n!} < eps$$

התוכנית תדפיס את הנתונים הבאים: x, eps, ואת e^x כפי שיחושב על ידה ואת מספר האיברים בטור שיחושב.

7. התכונה הבאה ידועה כנכונה עבור מספרים שלמים בתחום 1 עד 100: מכל מספר X ניתן לפרוש סדרה ...x_1, x_2 כמפורט להלן:

$x_1 = x$

$x_2 = \begin{cases} x_1/2 & \text{אם } x_1 \text{ זוגי} \\ 3x_1+1 & \text{אם } x_1 \text{ אי זוגי} \end{cases}$

$x_{n+1} = \begin{cases} x_n/2 & \text{אם } x_n \text{ זוגי} \\ 3x_n+1 & \text{אם } x_n \text{ אי זוגי} \end{cases}$

האיבר האחרון בסדרה יהיה המספר 1.

לדוגמה:

```
10,5,16,8,4,2,1
11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
```

יש לכתוב תוכנית, אשר תבצע את הפעולות הבאות:

א. תקרא נתון, שהוא מספר שלם.

ב. תבדוק אם הנתון הינו בתחום 1-100 (ותדפיס הודעת שגיאה אם לא בתחום).

ג. תדפיס את הסידרה שנפרשה מהמספר.

ד. תדפיס את האיבר הגדול ביותר בסדרה שנפרשה.

הרץ את התוכנית עם הנתונים הבאים: 10, 5-, 99, 102, 0.

הערה: שים לב שהשורה הראשונה בתוכנית c5e7 צריכה להיות `int x=0, num;`

8. כתוב תוכנית שתקבל מספרים שלמים ותדפיס אותם. לידם היא תשרטט קו, שיורכב מהתו שערך ASCII שלו הוא 233 (DFh) ואורכו יחסי לערך המספר. את הקו תשרטט פונקציה שתקבל כערך את המספר (אורכו של הקו יהיה יחסי למספר הנתון). בראש התדפיס יופיע קו עם שנתות. התוכנית תעצור כאשר יתקבל המספר 0 (אפס).

דוגמה:

```
num: + + + + + + + +
3
    ■ ■ ■
5
    ■ ■ ■ ■ ■
```

9. כתוב פונקציה copies(chr, n) המדפיסה את התו chr מספר פעמים (n). השתמש בפונקציה זו בתוכנית המדפיסה משולש שווה שוקיים המורכב מתו כלשהו שנקלט. התוכנית תקלוט גם את מספר התווים שירכיבו את בסיס המשולש, תבדוק שאורך הבסיס יהיה מספר אי-זוגי ותדפיס הודעת שגיאה, אם תקבל מספר אחר. לדוגמה, עבור הקלט "A" ו-5 נקבל:

```
    A
  A A A
A A A A A
```

10. יש לכתוב תוכנית שתחשב את הערך של הביטויים הלוגיים הנתונים בקלט. (אין תשובה בתקליטור).

הקלט יהיה למשל, כבדוגמה הבאה:

```
~(5 >= 2) && (6 < 11);
 (11 == 11) || (0 > 2);
```

התוכנית תקלוט את הביטויים, תדפיס ביטוי מתורגם ואת ערכו הלוגי: אמת (true) או שקר (false).

המפתח לביטויי היחס:

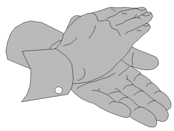
```
== EQ <= LE && AND
>  GT >= GE || OR
<  LT ~ NOT != NE
```

לדוגמה, עבור הביטוי הראשון נקבל את הפלט הבא:

```
> ~(5>=2)
not (5 GE 2)
> &&
AND
> (6<11)
(6 LT 11)
> ;
FALSE
```

הגדר את הפונקציות הבאות:

1. get_exp() - קולטת ביטוי לוגי, מדפיסה את תרגומו ומחזירה את ערכו הלוגי.
2. print_res() - מקבלת ערך לוגי ומדפיסה true או false בהתאם.
3. get_op() - קולטת את הפעולה הלוגית, מדפיסה את התרגום ומחזירה ערך שישמש לחישוב התוצאה הסופית. פונקציה זו תשמש גם להפסקת הקלט.



משימות:

תרגילים 4 - 5.

פרק 6

מערכים ומחרוזות

עד כה הכרנו והשתמשנו במשתנים ובקבועים בדידים. כדי להשתמש בערכו של משתנה היינו חייבים לפנות אליו בשמו המפורש. מאופיו של המחשב והמהירות בה מתבצעות הפעולות השונות ב-CPU, נגזרת היכולת לבצע סריקה מהירה על פני אזורי זיכרון גדולים ולהתייחס לערכיהם של משתנים רבים בו-זמנית.

נניח לדוגמה, שבאזור חיוג מסוים יש אלפיים מנויי טלפון וברצוננו לבדוק אם מספר טלפון מסוים שייך לאזור זה. לשם כך עלינו לבצע בתוכנית השוואה כנגד אלפיים משתנים שונים. התוכנית המתקבלת במקרה זה לא היתה שימושית בגלל גודלה וחוסר הגמישות שלה. אם נוסיף עוד עשרת אלפים מנויים חדשים, או משתנים, ניווכח שאין זה פתרון נכון.

השימוש בטכניקה של **מערכים** (arrays) מאפשר לבצע את המשימה בדרך יעילה, פשוטה וקצרה מבחינת התכנות. עושים זאת בעזרת הגדרה של רצף משתנים, הקרויים **איברים** (elements) ובעזרת לולאת תכנות פשוטה ה"מעבירה" אותנו על פניהם בעת ביצוע העיבוד. ניתן לראות את המערך כטבלה המכילה עמודה אחת בלבד, או שורה אחת של משתנים, ועל-כן נכנה אותה **מערך חד-מימדי** (Vector). כאשר טבלה זו תכיל עמודות נוספות נקבל מערך דו-מימדי, וכך גם ניתן ליצור **מערך רב-מימדי** (multi dimension array). **מחרוזת** (String) כוללת רצף של תווים, בדומה למערך חד-מימדי הכולל משתנים. לעיתים מקובל לכנות גם מערך חד-מימדי בשם Array.

בפרק זה נכיר מערך חד-מימדי, מחרוזת ומערכים רב-מימדיים ונלמד כיצד לאתחל מערכים אלה. גם נכיר חלק מהפונקציות הסטנדרטיות הכלולות בספריית המערכת המטפלות במחרוזות.

הגדרת מערך ואתחולו

כאשר עוסקים בהגדרת מערך, עלינו להפריד בין שני מקרים: המערך הסופי והמערך חסר המימדים. בניגוד לשפות עיליות אחרות, מאפשרת שפת C להגדיר מערך ללא מימדים ולהקצות עבורו מקום בזיכרון, על-פי מספר ערכי האתחול שלו. הגדרת מערך:

```
<טיפוס> <שם מערך> [<מספר איברים>];
```

או:

```
<טיפוס> <שם מערך> = {<ערכי אתחול>};
```

להלן מספר הגדרות קבילות למערכים:

```
short int Array[10];
int array[] = {100, 5, 7, 95};
```

מהו מערך?

זיכרון המחשב הוא סדרתי במבנהו. זיכרון מחשב בגודל של 256,000 בתים הוא למעשה סדרה רצופה בת 256,000 איברים בני בית אחד. בתוך סדרה זו ניתן להגדיר סדרה רציפה קטנה יותר. **המערך** (Array) הינו רצף של משתנים, שגודלם נקבע לפי הטיפוס שלהם.

ניקח לדוגמה מערך בשם my_array בן 10 איברים, אשר נגדיר אותו כך:

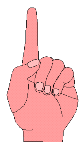
```
int my_array[10];
```

בפועל נקבל בזיכרון רצף של 10 משתנים מטיפוס שלם, אשר כל אחד מהם הוא בן 2 בתים, וביחד יידרשו למערך 20 בתים. אם תחילת המערך היא בכתובת 3000, נקבל את תמונת הזיכרון הבאה:

3001	איבר	ראשון	3000
3003	איבר	שני	3002
3005	איבר	שלישי	3004
3007	איבר	רביעי	3006
	...	...	
3017	איבר	תשיעי	3016
3019	איבר	עשירי	3018

שם המערך my_array הוא קבוע, וערכו (כתובתו) בדוגמה זו - 3000 (ערך זה תמיד יהיה כתובתו של האיבר הראשון במערך). מיקומו של המערך וערכו של שם המערך נקבעים בזמן תהליך ההידור. העובדה שאיברי המערך נמצאים בזיכרון ברציפות, מאפשרת פנייה נוחה באמצעות **אינדקס** (Index). איברי המערך array הם:

```
my_array[0], my_array[1]........ my_array[9]
```



שים לב!

*הפנייה לאיברים היא "**מבסיס אפס**", כלומר האינדקס של האיבר הראשון הוא **0**. טעות נפוצה היא לציין את האיבר הראשון בסימן הסידורי 1. כאשר פונים לאיבר מסוים במערך מתורגמת פנייה זו על ידי המהדר למספר פעולות קצרות ויעילות שתפקידן להגיע לכתובת המתאימה בזיכרון. הבסיס לפעולות אלו היא כתובת תחילת המערך, וגודל כל איבר בו.*

פנייה ל-my_array[4] תתורגם על ידי המהדר לביטוי מהצורה הבאה (שים לב שזהו האיבר **החמישי** במערך):

(my_array כתובת**) + 4 * (int** גודל הטיפוס**)**

לפי הדוגמה הקודמת תהיה הכתובת של האיבר החמישי:

```
3000 + 4 * 2 = 3008
```

הקצאת המקום בזיכרון נעשית על פי טיפוס המערך ומימדיו. בשפת C מוכרים טיפוסי המערכים כטיפוסי המשתנים. לדוגמה:

```
char letters[22];
float integral_vals[100];
double  big_array[15];
```

במקרה של המערך letters, יוקצו 22 בתים בזיכרון, מכיון שגודל הטיפוס תו הוא בית אחד בלבד. בהנחה שגודל הטיפוס double הוא 8 בתים, יהיה המערך big_array בן 120 בתים.

אתחול של מערך חד-מימדי

כדי לאתחל מערך ניתן להשתמש בלולאה או בהצבה מרוכזת. אתחולו של המערך my_array[] בערך אפס יכול להתבצע על ידי התוכנית הבאה:

```
void main(void)
{
  int my_array[10], i;
  for (i = 0; i < 10 ; ++i)
    my_array[i]=0;
}
```

מערך זה אפשר לאתחל בדרך שונה:

```
void main(void)
{
  int my_array[10] = {0,0,0,0,0,0,0,0,0,0};
}
```

את המערך letters נאתחל באופן הבא:

```
void main(void)
{
  char letters[26] = {'a','b','c','d',...,'x','y','z'};
}
```

בהנחה שהמערך letters נמצא בזיכרון בכתובת 4000 מתקבלת תמונת הזיכרון הבאה:

4000	4001	4002	4003	...	...	...	4010	...	...
65	66	67							
						85	86		
4011	4012	...	...	...	...	4020	4021	...	4026

הערכים שבתאים הם ערכי ASCII של האותיות "a" עד "z".

דוגמאות למערכים חד-מימדיים

הזכרנו קודם את הבעיה של אחסנה וטיפול באלפיים מספרי טלפון השייכים לאזור חיוג מסוים. נניח, שעלינו לאפשר את ביצוע הפעולות הבאות: הכנסת מספר חדש וחיפוש האם קיים. לשם כך נגדיר שתי פונקציות:

א. שם: insert()

פרמטר: num מטיפוס long.

ערך מוחזר: אין

ב. שם: exist()

פרמטר: num מטיפוס long.

ערך מוחזר: 1- אם המספר קיים / 0- אחרת — מטיפוס תו

עתה נכתוב את התוכנית. תחילה נגדיר את המערך, אחר-כך את הפונקציות insert() ו-exist(), ולבסוף נכתוב את התוכנית הראשית main(). בתוכנית זו נחפש מספר מבוקש ונציין אם קיים, או לא. לסיכום נציין את מספר האיברים הקיימים בפועל במערך.

```
      /*****      chp0601.c :קובץ תקליטור       *****/

      #include <stdio.h>

(1)   long  phone_array[2000] ;
(2)   int  current=0;

(3)   void insert (long num)
(4)   {
(5)     if (current<2000)
(6)       phone_array[current++] = num ;
(7)     else
(8)       printf("Can't insert number - no place left  \n");
(9)   }

(10)  char exist(long num)
(11)  {
(12)    int  I;

(13)    for(I=0; (I<current) && (phone_array[I]!=num);
                  ++I);

(14)    return(I<current) ;
(15)  }
```

```
(16)  void main(void)
(17)  {
(18)    long p_num=241051;

(19)    insert(226878) ;
(20)    insert(338533) ;
(21)    insert(241051) ;
(22)    insert(923845) ;

(23)    if (exist(p_num))
(24)      printf("The phone number %ld exists\n",p_num) ;
(25)    else
(26)      printf("The phone number %ld does not
                               exists\n",p_num) ;

(27)      printf("Amount of phone numbers is
                               %d\n",current) ;
(28)  }
```

הסבר:

א. בשורות 2-1 הגדרנו מערך גלובלי []phone_array שתכולתו המקסימלית אלפיים מספרי טלפון. הגדרנו גם משתנה current המורה את מספר התא הפנוי הבא במערך זה.

ב. הפונקציה ()insert בודקת את הערך הנוכחי של מספר התא הפנוי (שורה 5). אם ערך זה קטן מאלפיים, ניתן להכניס את המספר num ולקדם את ערכו של המשתנה. אחרת, תודפס הודעת אזהרה וערכו של num יאבד.

ג. בפונקציה ()exist השתמשנו בקיצורים שלמדנו כבר. בשורה 13 מבוצעת לולאת for, שתנאי עצירתה הוא שילוב של התנאים (I < current) וגם אי-שוויון של המספר הנוכחי והמספר הרצוי.

ד. האינדקס הזמני I מקודם בפעולת pre_increment בתוך לולאת for.

ה. הביטוי המוחזר בשורה 14 הוא למעשה ביטוי בוליאני. אם לא הגענו למקום התא החופשי, התנאי (phone_array[I] != num) לא מתקיים, כי האינדקס I עדיין קטן מהגבול current. כלומר, מתקיים ההיפך: phone_array[I] == num, והערך המוחזר (true) אכן מתאים למקרה.

ו. שים לב לפשטות וליעילות הפונקציות ()insert ו-()exist. בעזרת משפט בודד אפשר לבצע סריקה מהירה על פני אוסף גדול מאוד של ערכים הנמצאים במערך אחד.

תוצאת הריצה של התוכנית main תהיה:

```
The phone number  241051 exists
Amount of phone   numbers is 4
```

למערכת שיצרנו יש חיסרון בולט. כדי לתת תשובה לשאלת ()exist עלינו לסרוק את כל המערך. זהו תהליך שגוזל זמן רב כאשר כמות המספרים גדולה. האם ניתן לקצר תהליך זה? נמיין תחילה את המערך הקיים []phone_array, כדי שנוכל לאתר את מספר הטלפון המבוקש בהסתמך על עובדה זו.

לדוגמה, המערך הלא ממוין מכיל שלושה מספרי טלפון:

```
phone_array[0] = 263442
phone_array[1] = 226878
phone_array[2] = 241051
```

לאחר המיון נקבל במערך הממוין את הערך הקטן ביותר באיבר הראשון ואת הערך הגבוה ביותר באיבר האחרון.

```
Phone_array[0] = 226878
phone_array[1] = 241051
phone_array[2] = 263442
```

כעת אנו רוצים לבדוק האם המספר 235065 קיים במערך. נפנה למערך הממוין ונראה שהאיבר הראשון מכיל את המספר 226878. זהו מספר קטן מהמבוקש, ולכן נמשיך בחיפוש. האיבר הבא מכיל את המספר 241051, שהוא גדול מהמבוקש ולכן אין טעם להמשיך בחיפוש. תשובת ()exist במקרה זה תהיה false. ניתן לראות, שעבור מערך שגודלו אלפי איברים, מנגנון חיפוש זה יעיל בהרבה מהמנגנון הקודם, אם כי גם אותו ניתן לשפר. נדגים כעת שיטת מיון פשוטה ונרחיב אחר-כך את התוכנית, כדי שתכניס מספר חדש ותחפש מספר מבוקש.

```
      /*****      chp0602.c :קובץ תקליטור       *****/

      #include <stdio.h>

(1)   long  phone_array[2000] ;
(2)   int  current=0;

(3)   void sort (void)
(4)   {
(5)     long  temp;
(6)     int  I, j ;

(7)     for(I=0 ; (I<current) ; ++I)
(8)       for (temp=phone_array[I],j=I+1; j<current ; ++j)
(9)          if (phone_array[j]<temp)
               {
(10)           phone_array[I]=phone_array[j];
(11)           phone_array[j]=temp;
(12)           temp=phone_array[I];
(13)           }
(14)  }
```

```
(15)  void insert(long num)
(16)  {
(17)    if (current<2000)
(18)      {
(19)      phone_array[current++] = num ;
(20)      sort();
(21)      }
(22)    else
(23)    printf("Can't insert number - no place left \n");
(24)  }

(25)  char  exist (long num)
(26)  {
(27)    int  I;

(28)    for (I=0; (I<current) && (phone_array[I]<num);
                                              ++I);

(29)    return(phone_array[I]==num) ;
(30)  }

(31)  void main(void)
(32)  {
(33)    long p_num=241051;
(34)    int I;
(35)    insert(226878) ;
(36)    insert(338533) ;
(37)    insert(241051) ;
(38)    insert(923845) ;
(39)    if (exist(p_num))
(40)      printf("The phone number %ld exists\n",p_num) ;
(41)    else
(42)      printf("The phone number %ld does not
                                exists\n",p_num) ;

(43)      printf("Number of phones is %d\n",current) ;

(44)    for(I=0 ; I<current ; ++I)
(45)      printf("%ld,", phone_array[I]);

(46)  }
```

הסבר:

א. בשיטת המיון המוצגת בשורות 14-3 דואגים בכל שלב ושלב, שבתחילתו של תת-המערך יהיה האיבר הקטן ביותר מבין הנותרים. תת-מערך הוא המערך המתחיל באיבר I ומסתיים באיבר current. מכיון שערכו ההתחלתי של I הוא אפס - מתבצעת סריקה על פני כל המערך. בכל סיום שלב, האיבר הראשון הוא הקטן ביותר בחלק המערך עליו מבוצעת הסריקה.

ב. אם נתקלים באיבר קטן יותר מהאיבר שבתחילת תת-המערך (התנאי הבוליאני בשורה 9 מתקיים) תתבצענה השורות 12-10 ובמסגרתן יוחלפו שני האיברים באמצעות המשתנה temp.

ג. שתי לולאות for (בשורות 7 ו-8) מאפשרות סריקה מהירה ויעילה על פני תת-מערכים שאורכם הולך וקטן עד לסיום המיון.

ד. דיון נוסף בנושא המיון והסברים מפורטים לשיטות המיון תמצא לאחר הסבר זה, בתרגילים שבסוף הפרק ובנספח המשימות.

ה. לפונקציה ()insert נוספה הקריאה לפונקצית המיון (שורה 20). כאשר עוסקים במערך גדול, זוהי ירידה ניכרת ביעילות תהליך ההכנסה של מספר נוסף, מכיון שתהליך המיון מסובך וארוך. אנו מניחים כאן, שפעולות החיפוש ()exist רבות פי כמה מפעולות הכנסה, ומשום כך גדלה היעילות הכוללת של המערכת שבנינו.

ו. בשורה 28 ניתן להבחין שתנאי העצירה בפונקציה ()exist השתנה וכעת נצא מלולאת for, כאשר המספר שבדקנו גדול או שווה למספר המבוקש. כזכור, כאשר המערך אינו ממוין, יש לסרוק עד קצה המערך כדי להחליט שאכן המספר אינו קיים.

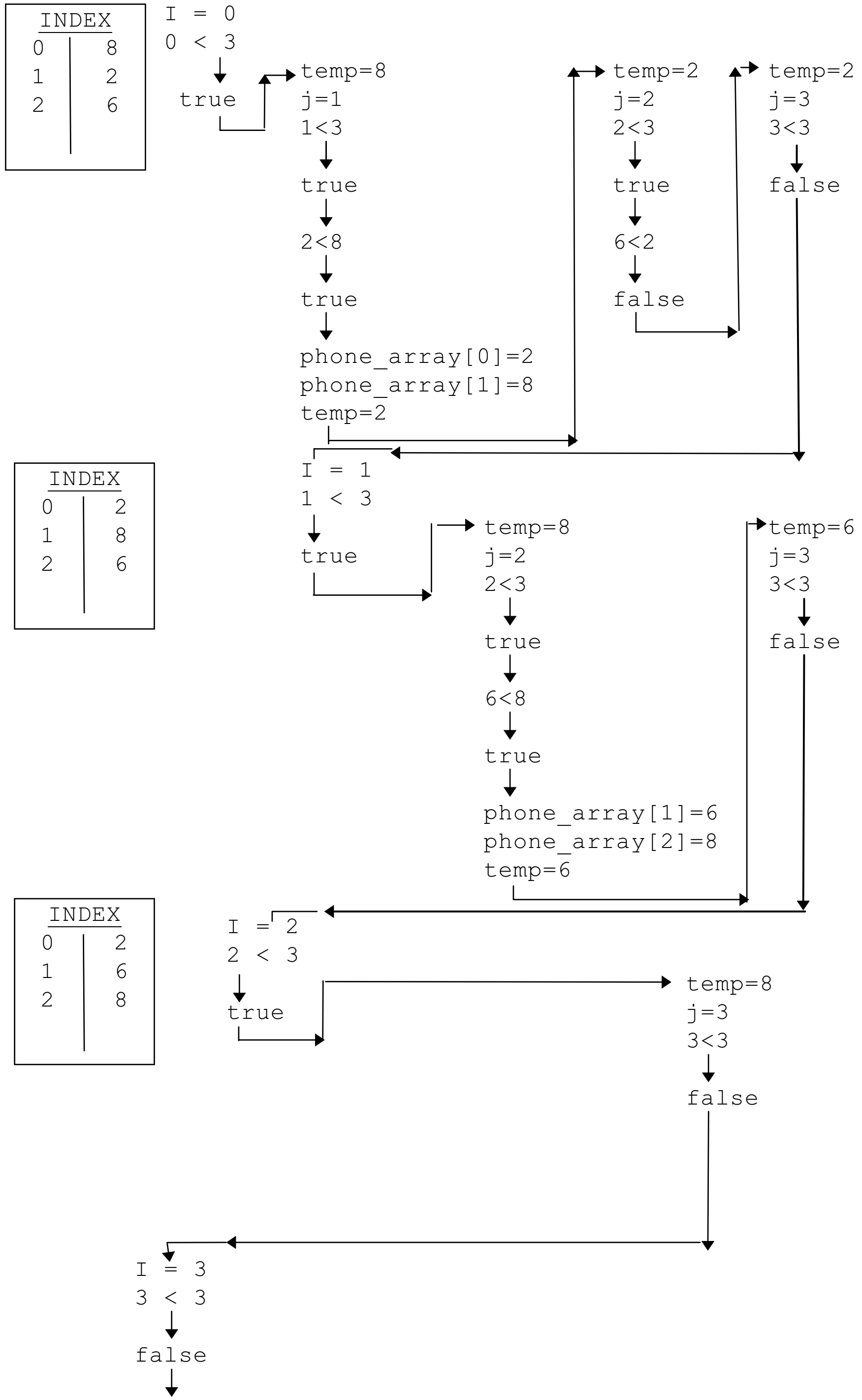
ז. הביטוי הבוליאני בשורה 29 השתנה בהתאם לתנאים שעלינו לבדוק.

בתרשים 6.1 מוצגת דוגמה של הפעלת ()sort כאשר המערך []phone_array מכיל שלושה איברים :

```
phone_array[0] = 8
phone_array[1] = 2
phone_array[2] = 6
```

התהליך המוצג בתרשים הינו מהלך מיון במחשב, אשר בסיומו יכיל המערך []phone_array את האיברים : 2, 6 ו-8 כשהם ממוינים בסדר עולה.

האיבר current מכיל את הערך 3.



תרשים 6.1: *מיון שלושה איברים במערך*

מחרוזות

המחרוזת (String) הינה מערך חד-מימדי של תווים (איברים מטיפוס char). כלל מקובל בשפת C הוא, שמחרוזת מסתיימת באיבר שערכו **אפס** (Null). השימוש במחרוזות בשפת C מקובל מאוד, וחלק ניכר מפעולות קלט/פלט של תוכנית נעשה באמצעותן.

בסעיף זה נדון במשפחה של מחרוזות, אשר מוגדרות באמצעות מערכים בעלי מימד סופי. כלומר, שידוע לנו מראש מספר האיברים שלהן. נכיר חלק מפונקציות הספריה הסטנדרטיות של שפת C העוסקות במחרוזות ונשתמש בהן בתוכניות. מטרתנו להגיע לבסוף לפונקציות היעילות והמצומצמות ביותר שמאפשרים הכלים שבידינו.

כיצד נגדיר מחרוזת?

```
Char <שם> [מספר תווים];
```

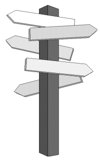
תחילה עלינו לדעת מהו האורך המקסימלי הדרוש לנו, למשל:

```
char str_one[20], string[20];
```

בדוגמה זו הגדרנו שתי מחרוזות, שכל אחת מהן מכילה **עד** 20 תווים. האורך שלהן בפועל ייקבע על ידי האתחול או ההצבה. מכיון שכאן לא בוצע אתחול, אורך המחרוזות האלו **אינו מוגדר**. אם ההגדרה הזו גלובלית, אורכן המעשי הוא אפס, שכן אתחול מערך גלובלי הוא אפס בכל האיברים, וכפי שציינו, "אפס" מסיים מחרוזת.

נגדיר מחרוזת נוספת ונאתחל אותה על ידי הצבה של ערכים כלשהם:

```
char msg[30] = "I am a string";
```



נציין מספר עובדות!

- *ניתן לאתחל מערך תווים בצורה שונה מאתחול מערך רגיל.*
- *את המחרוזת הזו אפשר לאתחל גם באופן הבא ('b' מציין תו רווח):*

```
char msg[30] = {'I', 'b', 'a', 'm', 'b', 'a', 'b', 's',
                                    't', 'r', 'I', 'n', 'g', 0};
```

- *ה-0 המסיים מחרוזת, אינו מופיע בצורה מפורשת בשיטת האתחול הראשונה, מכיון שהמהדר דואג לסיימה באפס.*
- *ניתן לאתחל מחרוזת ללא ציון גודלה. גודלה ייקבע על פי אורכה בפועל.*

```
Char msg[]="string";
```

כיצד ניתן לאתחל את המחרוזות הקודמות: str_one ו-string? לפניך קטע תוכנית המעתיק את ערך המחרוזת msg לשתיהן:

```
(1) int  I;
(2) for (I = 0; msg[I]; ++I)
(3) str_one[I] = string[I] = msg[I];
(4) str_one[I] = string[I] = 0;
```

הסבר:

א. ההעתקה מבוצעת איבר אחר איבר עד שמגיעים ל-0 המסיים.

ב. מכיון שהערך המסיים הוא אפס, הרי שהביטוי msg[I] יקבל **ערך שקר** (false) כאשר I יגיע אליו.

ג. בשורה (4) אנו דואגים לסיים את str_one ו-string גם כן ב-0.

פונקציה כללית המשמשת להעתקת מחרוזת אחת לאחרת נקראת strcpy(). לדוגמה:

```
    /*****      chp0603.c  :מתוך קובץ תקליטור       *****/

    #include <stdio.h>

(1) strcpy(char target[], char  source[])

(2) {
(3)   int  I;

(4)   for (I=0 ; target[I] = source[I] ; ++I)
        ;
(5) }
```

הסבר:

א. בשורה 1 מוגדרים הפרמטרים הפורמליים source ו-target. הגדרנו אותם כמערכים חסרי מימד ולמעשה הכרזנו עליהם כמערכים. המהדר יתייחס להצהרה זו במהלך תרגום הפונקציה, מתוך הנחה שהמערכים עצמם (האקטואליים) וגודלם, מוגדרים כמערכים בתוכנית הקוראת. יכולת הכרזה זו היא שמקנה לפונקציה את תכונת הכלליות.

ב. בדומה לקטע התוכנית הקודם מבוצעת ההעתקה איבר אחר איבר.

ג. לביטוי ההצבה שבמשפט for יש ערך, כפי שלמדנו בפרק העוסק במבנה התוכנית. ערך זה הוא ערך האיבר source[I]. כאשר נגיע לתו null נעצור, כיון שאז ערך הביטוי יהיה שקר.

ד. שים לב, שבשיטה זו ה-null מועתק קודם ל-target ומסיים אותו - ורק לאחר מכן משוערך התנאי הגורם לסיום לולאת for.

פונקציות ספריה לטיפול במחרוזות

כעת נכיר מספר פונקציות מחרוזות שימושיות נוספות הנכללות בספריה הסטנדרטית של שפת C, ובהמשך נציג אותן בליווי דוגמאות.

Strlen(str) מקבלת מחרוזת str ומחזירה ערך שלם, המציין את אורך המחרוזת על-פי המקום של התו המסיים אותה.

strcmp(str1,str2)	מקבלת שתי מחרוזות str1 ו-str2 ומשווה ביניהן באופן **לקסיקוגרפי**. היא מחזירה: ❖ ערך שלילי אם str1 קטנה מ-str2. ❖ אפס אם str1 שווה ל-str2. ❖ ערך חיובי אם str1 גדולה מ-str2
strcat(str1,str2)	מקבלת שתי מחרוזות str1 ו-str2 ומצרפת את str2 לסופה של str1.
strcpy(t,s)	העתקת המחרוזת s למחרוזת t, אותה הכרנו בדוגמה הקודמת.

הפונקציה ()strlen

```
        /*****      chp0603.c  :מתוך קובץ תקליטור       *****/

(1)   strlen( char str[])
(2)   {
(3)     int  i;

(4)     for (i=0 ; str[i] ; ++i)
           ;
(5)     return(i);
(6)   }
```

הסבר:

א. הפונקציה ()strlen עוברת באופן סדרתי על איברי המחרוזת ומגדילה תוך כדי כך את ערך המונה i.

ב. העצירה מתבצעת כאשר מגיעים לערך 0 (false) שהוא ה-0 המסיים (null).

דוגמה:

```
char msg[40] = "I am a string";
printf("The length of msg is %d\n", strlen(msg));
```

הפלט הוא:

```
The length of msg is 13
```

הפונקציה ()strcmp

```
    /*****      chp0603.c :מתוך קובץ תקליטור       *****/

(1)   strcmp(char str1[], char str2[])
(2)   {
(3)     int  i;

(4)     for (i=0 ; str1[i] == str2[i] && str1[i] &&
                   str2[i]; ++i)
          ;
(5)     return(str1[i]-str2[i]);
(6)   }
```

הסבר:

א. השוואה לקסיקוגרפית היא השוואה של ערכי התווים במחרוזות לפי סדר ה-א״ב.

בייצוג ASCII - ההשוואה בין התו ״a״ לבין התו ״c״ היא למעשה השוואה בין הערך 65 לבין הערך 67 המייצגים את התווים בהתאמה.

ב. אם המחרוזות אינן שוות, התנאי הראשון בשורה 4 לא יתקיים ולולאת for תיפסק.

ג. אם אחת המחרוזות קצרה מהשנייה, תיפסק לולאת for בעקבות התנאים הנוספים. הלולאה תיפסק מאותה סיבה גם כאשר המחרוזות זהות ונגיע בשתיהן ל-null המסיים.

ד. בשורה 5 מוחזר ההפרש בין התווים של שתי המחרוזות במקום בו נעצרה לולאת ה-for. במקרה של שוויון, מגיעים בשתיהן ל-null המסיים ולכן, התוצאה המוחזרת תהיה 0 (כלומר שוויון).

בדוק בעצמך את נכונות התוצאה במקרים האחרים עד שתשתכנע, שאכן הפונקציה פועלת כהלכה.

דוגמאות הרצה:

המחרוזות:

```
printf("Compare %s /%s => %d\n", "I am", "He is",
       strcmp("I am", "He is"));
```

התוצאה:

```
Compare I am   / He is => 1
```

המחרוזות:

```
strcmp("I am", "I don't");
```

התוצאה:

```
Compare I am  / I don't => -3
```

המחרוזות:

```
strcmp(msg,string);
```

התוצאה: בהנחה שהמערך []string מכיל עותק מדויק של []msg.

```
Compare I am    string / I am a string => 0
a
```

הפונקציה ()strcat

```
      /*****      chp0603.c  :קובץ תקליטור       *****/

(1)   strcat(char str1[], char str2[])
(2)   {
(3)     int  i, j;

(4)     for (i=0 ; str1[i] ;  ++i)
         ;
(5)     for (j=0 ; str1[i]=str2[j] ; ++i, ++j)
         ;
(7)   }
```

הסבר:

א. בשורה 4 עוברים באופן סדרתי על פני המחרוזת הראשונה במטרה להגיע לסופה.

ב. בשורה 5 מועתק תוכן המחרוזת str2, איבר אחר איבר, החל מסופה של str1 ואילך.

ג. כדי שהפעלת הפונקציה ()strcat לא תגרום לפגיעה בערכי זיכרון או משתנים, יש לדאוג לכך שהמערך str1 יהיה גדול דיו, כדי להכיל את המחרוזת המשולבת.

דוגמה:

```
strcat (msg, ", now I am two !!!");
printf("result of strcat() : %s \n",msg) ;
```

```
result of strcat()   : I am a string, now I am two!!!
```

השימוש ב-Const

כאשר מגדירים מערך ואין כל כוונה לשנות את תוכנו, אפשר להגדירו כ-const, כך שלא נוכל לשנותו גם אם נרצה, כי המהדר לא יאפשר לנו לעשות זאת.

נשכתב את הפונקציה strcmp שראינו בפיסקה קודמת, ונגדיר את מערכי המחרוזות כ-const. הפונקציה אינה אמורה לשנות את ערכם של מערכי המחרוזות, ולכן הגדרת const היא אבטחה נוספת שלא נשנה מערכים אלה בלי לשים לב.

```
strcmp (const char str1[], const char str2[])
{
   int i;

   for (i=0; str1[i] == str2[i] && str1[i] && str2[i]; i++)
      ;
   return (str1[i] - str2[1]);
}
```

מערכים רב-מימדיים

עד כה דנו במערכים חד-מימדיים. כדי לייצג טבלאות, מטריצות או מבנים מורכבים דרושה גם אפשרות להגדיר ולהשתמש במערכים בעלי יותר ממימד אחד. שפת C מאפשרת הגדרת מערכים כאלה, ללא כל הגבלה על מספר המימדים. אתחול וניצול של מערכים רב-מימדיים הינן פעולות מורכבות יותר מאשר במערכים חד-מימדיים ועל-כן דורשות תשומת לב רבה יותר מצד המשתמש. כדי להימנע מטעויות ולפשט פעולות אלו, נביא להלן מספר כללים בסיסיים חשובים.

בהגדרת מערך חד-מימדי, יש לציין את המספר המקסימלי של האיברים במערך. הגדרת מערך רב-מימדי נעשית, באופן אנלוגי, על ידי ציון הגודל המקסימלי של כל מימד. טבלה למשל, הינה מערך דו-מימדי שיש בו שורות ועמודות ומגדירים אותה באמצעות שני פרמטרים, הראשון עבור השורות, והשני - עבור העמודות. לדוגמה, נגדיר טבלה של שלמים בעלת 10 שורות, אשר בכל אחת מהן 4 עמודות:

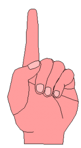
```
int table[10][4];
```

קוביה הינה דוגמה למערך תלת-מימדי, אשר אפשר להציג אותה כערימת שכבות דו-מימדיות המונחות זו על גבי זו, כאשר המימד השלישי מציין את הגובה. נגדיר עתה מערך רב-מימדי, אשר אפשר לראותו כמקבץ של טבלאות הצמודות זו לזו. "עובי" שכבת הטבלאות הדו-מימדיות יוצר את המימד השלישי. בדוגמה זו, "הטבלה" היא בת 5 שורות ו-100 עמודות ובת 8 שכבות.

```
int box_of_ints[8][100][5];
```

כדי להדפיס את האיבר שנמצא בעמודה 28, בשורה 3 שבטבלה **השנייה**, נרשום:

```
printf("%d\n", box_of_ints[1][27][2]);
```



זכור!

האיבר הראשון במערך מסומן ב-0 (אפס).

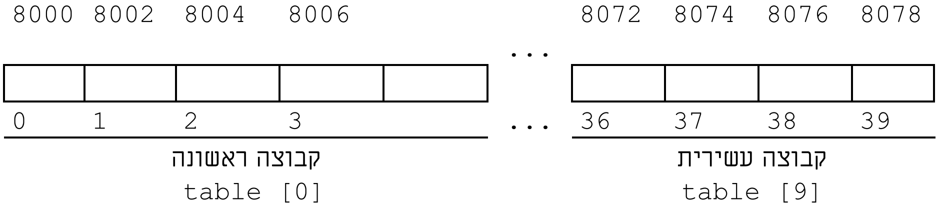
ברוב המקרים אין צורך במימדים נוספים, אף כי שפת C מאפשרת זאת. הטיפול בגופים כאלה אנלוגי לטיפול במערכים עד 3 מימדים. נציג למשל, את מהלך הטיפול של שפת C במערך רב-מימדי:

א. המימד החשוב ביותר הוא המימד **האחרון**, שהוא **הימני ביותר**. כאשר מגדירים מערך דו-מימדי כדוגמת

```
int table[10][4];
```

שומר המהדר 40 מקומות עוקבים בזיכרון עבור המספרים השלמים **ומצביע** עליהם באמצעות הקבוע table.

החלוקה הפנימית שעל פיה תבוצענה פעולות האתחול ומציאת ערך כלשהו של איבר במערך נקבעת על פי המימד השני, 4. באופן דמיוני ניתן לחלק את השורה הארוכה של 40 מקומות בזיכרון ל-10 קבוצות קטנות ועוקבות בנות 4 איברים כל אחת. הסכימה הבאה מתארת את המקרה הזה בהנחה שערך table הוא 8000.



ב. אתחול המערך הרב-מימדי יתבצע בהתאם לחלוקה לקבוצות. כל תת-קבוצה תאותחל בנפרד בתוך סוגריים מסולסלים, כפי שמוצג בדוגמה הבאה:

```
int table[10][4] =  {
                    {1,2,3,4},
                    {5,6,7,8},
                    ...
                    {33,34,35,36},
                    {37,38,39,40},
                    } ;
```

אם לא נציין אחת מתת-הקבוצות לא יאותחלו איברי אותה קבוצה. מאופי המבנה של המערך נגזרת עובדה נוספת - אפשר לאתחל מערך גם בדרך זו.

```
int table[10][4]={1,2,3,4,5,6...37,38,39,40};
```

דוגמה נוספת לאתחול איברי מערך רב-מימדי:

```
int table[10][4] =  {
                    {1,2},
                    {3,4},
                    ...
                    {19,20}
                    } ;
```

פעולה זו תאתחל רק את הזוג **הראשון** בכל **תת-קבוצה**.

נציג עתה שתי דוגמאות לשימוש במערכים דו-מימדיים לאתחול של לוח-הכפל והדפסת כפולה מתוכו.

א. אתחול מהיר של מערך דו-מימדי, עבור לוח הכפל: הפונקציה ()init.

ב. הדפסה של מכפלה על פי לוח הכפל: הפונקציה ()multiply.

```
        /*****      chp0604.c :קובץ תקליטור        *****/

#include <stdio.h>

int  squar[10][10];

void init(void)
{
   int i, j;

   for (i=0 ; i<10 ; ++i)
      for (j=0 ; j<10 ; ++j)
          squar[i][j] = i*j ;

}

multiply(int num1, int num2)
{
   if (num1>=10 || num2>=10)
      {
      printf("error - out of limits \n");
      return(-1);
      }
   else
      return(squar[num1][num2]);
}

void main(void)
{
   int n1, n2;

   init();

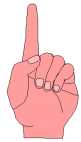
      for (n1=n2=-1 ; n1 ;)
          {
          printf("Enter two numbers:");
          scanf("%d%d", &n1, &n2);
          printf("\nResult: %d * %d = %d\n",
                     n1,n2,multiply(n1,n2));
          }
}
```

מיון וחיפוש במערכים

מיון (sort) היא פעולה של סידור איברים במערך על פי **מפתח** (key). מפתח יכול להיות סדר א-ב של שמות כמו בספר טלפונים, או מספרי תעודת זהות של תלמידים, למשל. מיון לפי תאריך או חתך בין תאריכים, כדי לבדוק את מצב חשבון הבנק ביום מסוים או בחודש מסוים.

מפתח יכול להיות כל ערך שהוא. לדוגמה, א-ב, תאריך, תעודת זהות, מספר קטלוגי, סכום כסף, גודל של קובץ ועוד.

מכאן אנו למדים שמיון הינו תהליך נחוץ ביותר: מה שנקרא "לעשות סדר בבלגן".



שים לב!

ניתן למיין טבלה על פי מספר מפתחות. לדוגמה, מיון מחלקות בחברה לפי מפתח מספרי מחלקות, ומיון שמות העובדים על פי מפתח א-ב או תעודת זהות בכל מחלקה ומחלקה.

מדוע צריך חיפוש?

נניח שיש לנו ספר טלפונים ממוחשב המכיל אלפי מנויים ואנו רוצים למצוא שם המתחיל באות 'ב'. קודמת לאות 'ב' היא האות 'א'. בהנחה שיש לנו 1,000 מנויים באות "א" נצטרך לדפדף (לעבור) על כל המנויים באות 'א' ולהמשיך לדפדף באות 'ב' כדי למצוא את שם המנוי שלנו. דבר זה מייגע, גוזל המון זמן ומיותר. מנגנון חיפוש אשר ניתן לתכנת ולמצוא שם בלחיצת כפתור לפי מפתח א-ב. למשל, כאשר נקיש את האות 'ב', בחלון קלט, מנגנון החיפוש יביא אותנו למנוי הראשון אשר שמו או שם משפחתו מתחיל באות 'ב'. חסכנו בכך דפדוף על גבי 1,000 שמות מנויים המתחילים באות 'א'. כאשר נקיש בחלון הקלט את שמו המלא של המנוי נקבל באופן מיידי את פרטיו או את פרטי כל המנויים העונים לשם דומה. זמן הדפדוף יתקרב ל-0, וזמן החיפוש תלוי ביעילות התוכנית. מדוגמה זו ניתן ללמוד שמנגנון חיפוש הינו חלק חיוני ביותר בכל מערכת.

עתה נבחן כמה שיטות למיון וחיפוש.

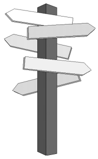
מיון בועות

מטרת **מיון בועות** היא לסדר את המספרים הנתונים במערך חד-מימדי באופן שבו נוכל לקרוא אותו מהספרה הקטנה ביותר עד לספרה הגדולה ביותר או ההיפך - מהספרה הגדולה ביותר עד לספרה הקטנה ביותר.

כיצד יתבצע מיון זה?

עוברים בלולאה על המערך ומשווים בין שני מספרים עוקבים. כאשר הראשון גדול מהשני מחליפים ביניהם. מכאן, שהמספר הקטן עלה שלב אחד למעלה, והגדול ירד שלב אחד למטה. בסוף הסיבוב הראשון נמצא בתא האחרון במערך את המספר הגדול ביותר במערך ונמצא גם שכלל המספרים הקטנים עלו בשלב אחד למעלה. האם המערך כבר ממוין? עדיין לא. לכן יידרש עוד סיבוב, אך הפעם נגיע אל התא האחד לפני האחרון מפני שבתא האחרון נמצא כבר המספר הגדול ביותר, בסיבוב הבא אחריו נגיע עד לתא האחרון פחות 2, מפני שבתא שלפני אחרון נמצא המספר שלפני הגדול ביותר.

מכאן נובע, שמספר הסיבובים יהיה כמספר התאים במערך. ניתן לקצר את מספר הסיבובים, אם נבדוק בכל סיבוב אם אכן החלפנו תא. אם לא החלפנו תא, קרוב לוודאי שהמערך כבר ממוין ואין צורך בסיבובי החלפה נוספים.



נקודת ציון!

*ניתן לראות, שהמספרים הקטנים הולכים ו"צפים" כלפי מעלה בדומה לבועות במים. מכאן נובע שם שיטת המיון הזו, "**מיון בועות**".*

בתוכנית הבאה נלמד כיצד למיין מערך בשיטת מיון בועות.

```
        /*****     chp0605.c : קובץ תקליטור      *****/

#include <stdio.h>
#include <conio.h>

#define LEN 10

void swap(int arr[], int i);
void printf_array(int arr[]);
/*
 ; -------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; --------------------------------------------------------*/
int main(void)
 {
  int arr[LEN] = { 3, 5, 0, 9, 1, 6, 8, 7, 2, 4 } ,i ,j;

  clrscr();
  printf("מיון מערך בשיטת מיון בועות\n");
  printf("המערך לפני מיון\n");
  printf_array(arr);
  /* תהליך המיון בשיטת "מיון בועות" */
  for (j = 0 ; j < LEN ; j++)
   {
```

```
  /*
    סריקת המערך עד לתא אחד לפני אחרון. מכיון, שבסוף הסיבוב הראשון בתא האחרון
    "ישב" המספר הגדול ביותר בסיבוב הבא אנו נסרוק את המערך עד לתא שני לפני התא
    אשר בו מאוחסן המספר הגדול ביותר. כיון שאנו בודקים מספרים בזוגות, יש לסרוק את
    המערך תמיד תא אחד פחות התא האחרון הרצוי.
  */
    for (i = 0 ; i < LEN - (j + 1) ; i++)
        if ( arr[i] > arr[i + 1] ) swap(arr, i);
   }
  printf("המערך אחרי מיון\n");
  printf_array(arr);
  printf("Press any key");
  getch();
  return(0);
 }
/* ----------------------------------------------------------
 ;  F U N C T I O N S
 ; --------------------------------------------------------*/
void swap(int arr[], int i)
 {
/*  תהליך החלפת המספרים במערך  */
  int tmp;

  tmp = arr[i];
  arr[i] = arr[i + 1];
  arr[i + 1] = tmp;
 }

void printf_array(int arr[])
 {
  /*  הדפסה סדרתית של המערך  */
  int i;

  for (i = 0 ; i < LEN ; i++)
     printf("arr[%d] = %2d\n", i ,arr[i]);
 }
```

תוצאות ריצת התוכנית:

```
מיון מערך בשיטת מיון בועות
המערך לפני מיון
arr[0] = 3
arr[1] = 5
arr[2] = 0
arr[3] = 9
arr[4] = 1
```

```
arr[5] = 6
arr[6] = 8
arr[7] = 7
arr[8] = 2
arr[9] = 4
```
המערך אחרי מיון
```
arr[0] = 0
arr[1] = 1
arr[2] = 2
arr[3] = 3
arr[4] = 4
arr[5] = 5
arr[6] = 6
arr[7] = 7
arr[8] = 8
arr[9] = 9
```

שים לב!

בשיטת מיון זו אין צורך להגדיר מערך נוסף, אלא תא עזר להחלפה בין תאי מערך. פעולה זו חוסכת מקום רב בזיכרון.

מיון בועות של מחרוזות נעשה באותה שיטה, אלא שכאן השוואה בין שני אלמנטים במערך נעשית על ידי פונקציית הספריה strcmp המחזירה:

(ערך קטן מ-0) 0 > → תוחזר מילה ב׳ < מילה א׳ בסדר לקסיקוגרפי (א-ב)

0 → תוחזר מילה ב׳ = מילה א׳

0 < → תוחזר מילה ב׳ > מילה א׳

החלפה בין שני אלמנטים (הפונקציה swap) תתבצע בעזרת פונקציית הספריה strcpy.

מיון מחרוזות בעזרת מערך אינדקסים

בתהליך מיון זה דרושה טבלה נוספת, **טבלת אינדקסים**, אשר אורכה כמספר השורות בטבלת המחרוזות. טבלה זו מכילה לפני המיון את האינדקסים של טבלת המחרוזות - כלומר, החל מ-0 ועד למספר השורות בסדר רץ (לדוגמה, אם יש 5 שורות האינדקסים יהיו: 0, 1, 2, 3, 4).

שיטת המיון היא "מיון בועות", אך כאן ההבדל הוא שהאינדקסים של טבלת המחרוזות נמצאים בטבלת אינדקסים נפרדת, והם מצביעים על סדר המיון. ההשוואות בין המחרוזות מתבצעות על פי האינדקסים שבטבלת האינדקסים.

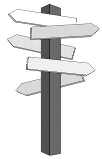
דוגמה

טבלת האינדקסים
```
int idx [5]={0,1,2,3,4};
```
טבלת המחרוזות
```
char str [5];
```

הפניה לטבלת המחרוזות תהיה:

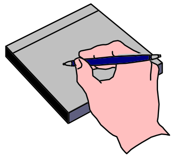
```
str [idx[0]] ;
```

תא idx[0] מכיל את המספר 0 ולכן נתייחס למחרוזת הראשונה בטבלה str.



נקודת ציון!

המיון מתבצע אך ורק בשינוי סדר המספרים בטבלת האינדקסים idx ועל פי סדר זה יוצגו המחרוזות.



הערה!

כיון שאיננו משנים דבר בטבלת המחרוזות, ניתן להגדירה כקבוע:

```
const char str[5] :
```

לפנינו תוכנית דוגמה המדגימה מיון מחרוזות בעזרת מערך אינדקסים.

```
        /*****  קובץ תקליטור: chp0606.c  *****/

#include <stdio.h>
#include <conio.h>
#include <string.h>

#define ROW 5

void swap(int idx[], int j);
void print_string(const char sStrs[ROW][30], int idx[]);

/* ----------------------------------------------------------
;  M A I N - P R O G R A M
 ; -----------------------------------------------------*/
int main(void)
 {
   /* כיוון שאיננו מתכוונים לשנות את מערך המחרוזות נגדירו כמערך, מחרוזות קבוע */
   const char sStrs[ROW][30] = {
                             "my Internet E-mail",
                             "boots_hs@netvision.net.il",
                             "boots_h soft",
                             "what is h",
                             "in a year or two from 1996",
                             };
                  /* מערך המכיל אינדקס של מילה במערך, אשר צריכה לבוא
                     במקום האינדקס sStrs[אינדקס] כשמערך המילים ממוין */
   int idx[ROW] = { 0, 1, 2, 3, 4 };
   int i, j;
```

```
  clrscr();
  printf("מיון מחרוזות בעזרת מערך אינדקסים\n");
  printf("המערך לפני מיון\n");
  print_string(sStrs, idx);
  /*  "תהליך המיון בשיטת "מיון בועות  */
  for (i = 0; i < ROW; i++)
   {
    for (j = 0; j < 5 - (i + 1); j++)
                        /*יש לשים לב שהאינדקסים במערך הם ערכי מערך
                           האינדקסים ומערך המחרוזות נשאר ללא שינוי   */
        if (strcmp(sStrs[idx[j]], sStrs[idx[j + 1]]) > 0)
         swap(idx, j);
   }
  printf("המערך אחרי מיון\n");
  print_string(sStrs, idx);
  printf("Press any key");
  getch();
  return(0);
 }

void swap(int idx[], int j)
 {
/* sStrs תהליך החלפת האינדקסים המצביעים על מילה במערך  */
  int tmp;

  tmp = idx[j];
  idx[j] = idx[j + 1];
  idx[j + 1] = tmp;
 }

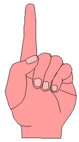
void print_string(const char sStrs[ROW][30], int idx[])
 {
/*   הדפסת המערך על פי האינדקסים המצביעים על אינדקס מילה  */
  int i;

  for (i = 0 ;i < 5 ; i++)
    printf ("sStrs[%d] = %s\n", i, sStrs[idx[i]]);
 }
```

תוצאות ריצת התוכנית:

```
מיון מחרוזות בעזרת מערך אינדקסים
המערך לפני מיון
sStrs[0] = my internet E-mail
sStrs[1] = boots_h@netvision.net.il
sStrs[2] = boots_h soft
```

```
sStrs[3] = what is h
sStrs[4] = in a year or two from 1996
המערך אחרי מיון
sStrs[0] = boots_h soft
sStrs[1] = boots_h@netvision.net.il
sStrs[2] = in a year or two from 1996
sStrs[3] = my internet E-mail
sStrs[4] = what is h
Press any key
```



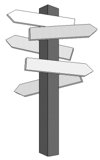
שים לב!

חסרונות שיטת המיון הם:

א. תופסת מקום בזיכרון עבור טבלת האינדקסים.

ב. מאוד לא נוחה כשרוצים להוסיף מילים חדשות, שכן יש לעדכן שתי טבלאות גם בתוספת מחרוזות ואינדקסים, גם במיון מחודש וגם בתוספת זיכרון.

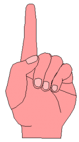
חיפוש בינארי בשיטת "אריה במדבר"



נקודת ציון!

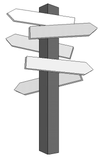
*החיפוש הבסיסי ביותר הוא "**חיפוש לינארי**" בהנחה שהמערך **ממוין**, מתחילים באיבר הראשון במערך ומתקדמים סדרתית לאיבר הבא, עד שנתקלים באיבר הרצוי. אם מגיעים לאיבר שערכו גדול מערך האיבר שמחפשים, פירושו של דבר שהאיבר הרצוי אינו נמצא.*

שיטה יעילה יותר לחיפוש איבר במערך היא שיטת ה**חיפוש הבינארי**. לפני שימוש בשיטה זו יש לדאוג למיון המערך. אפשר לערוך הקבלה של חיפוש זה לחיפוש מילה במילון ועל פי הקבלה זו נלמד את עקרון החיפוש הבינארי. לפנינו מילון ונתבקשנו למצוא את הפירוש למילה "עשיר". תחילה ננסה לפתוח את המילון באמצע - אם פתחנו במקרה בדף הנכון, נחפש את המילה ונסיים. נניח שפתחנו באות "ח", אנו יודעים ש-"ע" ("עשיר" - המילה המבוקשת) גדולה מ-"ח", ולכן אין כל סיכוי שנמצא את המילה עשיר בחצי הימני של המילון. בכך חסכנו 50% מזמן החיפוש בשיטת ה"חיפוש הלינארי", כי איננו צריכים לעבור ולבדוק את כל המילים בחצי הראשון של המילון. אנו יודעים כעת שהמילה "עשיר" נמצאת בחצי השמאלי של המילון, ולכן נפתח את החצי השמאלי באמצע. אם פתחנו בדף הנכון, נחפש את המילה ונסיים. אם שוב לא מצאנו, נבדוק שוב. נניח, שפתחנו באות "פ", ולכן אנו יודעים שהמילה "עשיר", נמצאת הפעם בחלקו הימני של המילון. ושוב נחלק חלק זה לחצי, עד שנמצא את המילה המבוקשת.



שים לב!

אין אנו מחלקים בכל פעם מחדש את כל המילון לחצי, אלא בפעם השנייה מחלקים את חצי המילון הימני או השמאלי לחצי. כך, שאנו ברבע, ואם נדרש, נחלק רבע זה בחצי וכך הלאה.



נקודת ציון!

*שיטת חיפוש זו נקראת גם "**אריה במדבר**".*

דוגמה

נחפש את המספר 10 במערך הבא: `6, 7, 8, 9, 10, 13, 21`

```
6    7    8    9  |  10    13    21  |
                           ↑
```

- ❖ נחלק את המערך לחצי.
- ❖ האם 9 הוא המספר המבוקש? לא.
- ❖ 10>9, לכן, נעבור לחלק הימני.

```
10    13    21
       ↑
```

- ❖ נחלק את החלק הימני לחצי.
- ❖ האם 13 הוא המספר המבוקש? לא.
- ❖ 10<13, לכן, נעבור לחלק השמאלי.

```
10
```

החלק השמאלי — האם 10 הוא המספר המבוקש? כן.

לפנינו תוכנית דוגמה המדגימה חיפוש בשיטת "אריה במדבר", חיפוש בינארי.

```
/*****       chp0607.c :קובץ תקליטור       *****/

#include <stdio.h>
#include <conio.h>

#define LEN 10
#define NOT_FOUND -1

void sort(int array[]);
void swap(int array[], int i);
void printf_array(int array[]);
int  binary_search(int array[], int var);
```

```
/* ---------------------------------------------------------
 ; M A I N - P R O G R A M
;--------------------------------------------------------*/
int main(void)
 {
   int array[LEN] = { 9, 3, 2, 4, 7, 6, 5, 8, 1, 10 };
   int var = 7;                          /*  האיבר אותו אנו רוצים למצוא  */
   int rv;

   clrscr();
   printf("המערך לפני מיון\n");
   printf_array(array);
                                              /*  תחילה יש למיין את המערך  */
   sort(array);
   printf("המערך אחרי מיון\n");
   printf_array(array);
   /*  בצע חיפוש  */
   printf("7 : האיבר, שאותו נחפש הוא");
   rv = binary_search(array, var);
   if (rv == NOT_FOUND)
         printf("\n האיבר לא נמצא במערך");
   else
         printf("\n%d : האיבר נמצא והוא יושב באינדקס", rv);
   printf("\nPress any key");
   getch();
   return(0);
 }
/* ---------------------------------------------------------
 ; F U N C T I O N S
 ; -------------------------------------------------------*/
void sort(int array[])
 {
                                       /*  תהליך המיון בשיטת "מיון בועות"  */
   int i, j;

   for (j = 0 ; j < LEN ; j++)
     for (i = 0 ; i < LEN - (j + 1) ; i++)
         if ( array[i] > array[i + 1] ) swap(array, i);
 }

void swap(int array[], int i)
 {
                                         /*  תהליך החלפת המספרים במערך  */
   int tmp;
   tmp = array[i];
```

```
   array[i] = array[i + 1];
   array[i + 1] = tmp;
 }

void printf_array(int array[])
 {
/* הדפסה סדרתית של המערך */
   int i;

   for (i = 0 ; i < LEN ; i++)
     printf("arr[%d] = %2d\n", i ,array[i]);
 }

int binary_search(int array[], int var)
 {
/* חיפוש בינארי בשיטת "אריה במדבר" */
   int first_idx = 0, middle_idx, last_idx = LEN - 1;

   while (first_idx <= last_idx)
     {
/* מציאת נקודת האמצע במערך */
     middle_idx = (first_idx + last_idx) / 2;
/* תחילה נבדוק האם מצאנו את הערך הרצוי */
     if (array[middle_idx] == var)
        return(middle_idx);
/* אם לא מצאנו נבדוק באיזה חלק של המערך יתכן ונמצא אותו ונעדכן את האינדקס */
        else if (array[middle_idx] < var)
/* האיבר נמצא בחלק הימני של המערך */
          first_idx = middle_idx + 1;
        else
/* האיבר נמצא בחלק השמאלי של המערך */
          last_idx = middle_idx - 1;
     }
   return(NOT_FOUND);
 }
```

תוצאות ריצת התוכנית:

```
arr[0] =  9
arr[1] =  3
arr[2] =  2
arr[3] =  4
arr[4] =  7
arr[5] =  6
arr[6] =  5
arr[7] =  8
```

```
arr[8] =  1
arr[9] = 10
המערך אחרי מיון
arr[0] =  1
arr[1] =  2
arr[2] =  3
arr[3] =  4
arr[4] =  5
arr[5] =  6
arr[6] =  7
arr[7] =  8
arr[8] =  9
arr[9] = 10
האיבר שאותו נחפש הוא: 7
האיבר נמצא והוא יושב באינדקס: 6
Press any key
```

אתחול מערכים בערכים אקראיים

לעיתים עלינו לציין מספר אקראי, או לאתחל מערך בערכים אקראיים. ערכים אקראיים יכולים להחליף קלט מידע לצורך בדיקת פונקציית מיון למשל, או לצורך כתיבת משחק ניחוש.

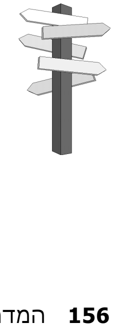
כיצד נבצע אתחול אקראי?

במהדר של שפת C קיים מנגנון המחולל מספרים אקראיים. עלינו לאתחל מחולל זה כדי ליצור מספרים אקראיים. תהליך יצירת המספר הוא תהליך חישובי.

כדי ליצור מספר אקראי שאינו חוזר על עצמו, יש לכלול בחישוב גורם המשתנה בכל פעם. גורם זה הוא זמן המערכת המשתנה בכל אלפית שנייה. זמן המערכת נכלל בחישוב, ולכן, נוכל לקבל סדרה ארוכה של מספרים השונים זה מזה.

ה**מאקרו randomize** מאתחל את המחולל במספרים אקראיים, כאשר אתחול זה מתבסס על זמן המערכת.

ה**פונקציה random(<מספר>)** מחזיקה מספרים אקראיים מטיפוס integer החל מהמספר 0 ועד למספר כלשהו, שהוא הפרמטר שהפונקציה מקבלת.



נקודות לציון!

כדי לקבל מספרים החל מ-1, יש להוסיף 1 לערך האקראי שהפונקציה מפיקה:

```
random(<מספר>) +1
```

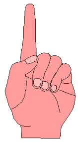
קיימת דרך נוספת לקבלת מספרים אקראיים אשר מקבלת נקודת התחלה, ומנקודה זו נקבל את המספרים האקראיים. מבנה הפונקציה:

```
time_t;
srand ((unsigned) time (&t));
```

time_t הינו מבנה המייצג זמן: שעות, דקות, שניות, אלפיות השנייה. גם פונקציה זו משתמשת במנגנון הזמן כדי לחולל מספרים אקראיים.

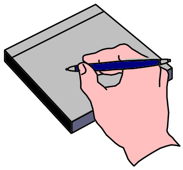
הפונקציה rand מחזירה מספרים שלמים מטיפוס integer. מבנה הפונקציה:

```
int rnd = (rand() %100) +1);
```



שים לב!

אם לא נוסיף 1 לפונקציה זו יתקבלו המספרים מ-0 ועד 99.
כאשר נוסיף 1 לפונקציה, נקבל את המספרים מ-1 עד 100.



הערה!

*לפונקציה **srand** יש מנגנון פנימי המחשב את המספרים האקראיים. כאשר נאתחל אותה בזמן המערכת בכל הרצה של התוכנית יתקבלו מספרים שונים. כלומר, זמן המערכת הוא מעין נקודת התחלה לחישוב, והרי זמן המערכת משתנה כל העת לכן יתקבלו מספרים שונים.*

כאשר נאתחל את הפונקציה במספר קבוע, נקבל סדרת מספרים אקראיים קבועים (ואין כאן סתירה). אותה סדרת מספרים תחזור על עצמה שוב ושוב מפני שנקודת ההתחלה של החישוב קבועה.

התוכנית הבאה ממחישה את אופן השימוש בפונקציות המחוללות מספרים אקראיים לצורך אתחול מערכים.

```
/*****      chp0608.c :קובץ תקליטור       *****/

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>  /* rand() -לצורך שימוש ב  */
   /* srand() לצורך שימוש במבנה זמן המערכת בפונקציה  */
#include <time.h>
#define LEN 5

void print_array(int array[], char title[]);
void rand_example(int array[]);
void random_example(int array[]);
void srand_example(int array[]);
```

```
/* -----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ;-----------------------------------------------------------*/
int main(void)
 {
   int array[LEN];

   clrscr();
   rand_example(array);
   print_array(array, "rand_example");
   random_example(array);
   print_array(array, "random_example");
   srand_example(array);
   print_array(array, "srand_example");
   printf("Press any key");
   getch();
   return(0);
 }
/* -----------------------------------------------------------
 ;  F U N C T I O N S
 ; ----------------------------------------------------------*/
void print_array(int array[], char title[])
 {
   int i;

   printf("\n%s\n", title);
   for (i = 0; i < LEN; i++)
         printf("array[%d] = %d\n", i, array[i]);
 }

void rand_example(int array[])
 {
               /*  המערך מקבל  ערכים אקראיים בין 1 ל- 100 במסגרת קבועה  */
   int i;

   for (i = 0; i < LEN; i++)
         array[i] = (rand() % 100) + 1;
 }

void random_example(int array[])
 {
               /*  המערך מקבל ערכים אקראיים בין 1 ל- 100 במסגרת משתנה  */
   int i;
```

```
  /*       randomize() - מאקרו המאתחל את מחולל המספרים האקראיים. אתחול זה
  מתבסס על שעון המערכת, כלומר הנוסחה המחשבת את המספרים האקראיים משתמשת
            בחישוביה בשעון המערכת. לכן המספרים בכל אתחול של המערך יהיו שונים  */
   randomize();
   for (i = 0; i < LEN; i++)
         array[i] = random(100) + 1;
 }

void srand_example(int array[])
 {
                    /*  המערך מקבל ערכים אקראיים בין 1 ל- 100 במסגרת משתנה  */
   int i;
                  /*  time_t  מבנה המיצג זמן שעות:דקות:שניות:אלפיות השנייה  */
   time_t t;
/* srand  –  אתחול מחולל המספרים האקראיים, אלא שכאן מחולל המספרים מקבל
     נקודת התחלה, שהחל ממנה נקבל את המספרים האקראיים. אם נקודת ההתחלה היא
     השעון, אז המספרים בכל אתחול של המערך יהיו שונים, אחרת, אותה סדרת מספרים
                                                       תחזור על עצמה.  */
   srand((unsigned) time(&t));
   for (i = 0; i < LEN; i++)
         array[i] = (rand() % 100) + 1;
 }
```

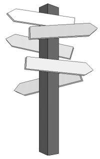
תוצאות ריצת התוכנית:

```
rand_example
arr[0] = 47
arr[1] = 31
arr[2] = 83
arr[3] = 91
arr[4] = 57

random_exapmle
arr[0] = 92
arr[1] = 49
arr[2] = 66
arr[3] = 59
arr[4] = 91
srand example
arr[0] = 55
arr[1] = 72
arr[2] = 16
arr[3] = 43
arr[4] = 70
Press any key
```

כיצד יתקבלו המספרים האקראיים?

מספר אקראי מתקבל על ידי נוסחה פשוטה, כאשר נקודת המפתח לחישוב הוא המשתנה seed אשר מתקבל על ידי התוכניתן.



נקודת ציון!

ערך המשתנה seed הוא בדרך כלל זמן המערכת.

הפונקציות הבאות ממחישות תהליך חישוב של מספר אקראי:

```
/*****     chp0609.c :קובץ תקליטור     *****/

#include <stdlib.h>

#define MULTIPLIER  0x015a4e35L
#define INCREMENT   1

static long Seed = 1L;

void srand(unsigned seed)
 {
              /* המשתנה seed מקבל בדרך כלל את זמן המערכת (השעה) */
  Seed = seed;
 }

int rand(void)
 {
  Seed = MULTIPLIER * Seed + INCREMENT;
  return((int)(Seed >> 16) & 0x7fff);
 }
```

תרגילים

תשובות בתקליטור: קבצים C:\tclite\targilim\c6e1-c6e11

1. כתוב תוכנית הקוראת את הערכים בווקטור, ומחליפה כל איבר בסכום כל האיברים הקודמים לו.
2. נתונה סדרה גדולה המורכבת מסדרות של מספרים ממשיים. כל סדרה מסתיימת ב-0 והסדרה המורכבת מסתיימת גם היא ב-0. כתוב תוכנית, אשר:

❖ תאתר את האיבר הגדול ביותר (מקסימום) בכל סדרה ואת המקום היחסי שלו בסדרה.

❖ תאתר את האיבר הגדול ביותר של כל הסדרות, ואת הסדרה שאליה הוא שייך.

❖ תאתר את הסדרה הקצרה ביותר ואת אורכה.

הרץ את התוכנית עבור הנתונים הבאים (כל סדרת מספרים מופרדת מחברתה בנקודה-פסיק):

```
5,1,10,-20,5,0;3.3,2.7,0;15,22,-10,7.5,-1,0;0;
```

3. נתונה מטריצה במבנה ריבוע קסם, שבו סכום המספרים שווה בכל שורה, עמודה ואלכסון.

לדוגמה, נתון ריבוע קסם, שבו סכום המספרים הוא 15:

6 1 8

7 5 3

2 9 4

כתוב תוכנית שמקבלת כקלט את מימדי הריבוע ואת ערכיו, ובודקת אם הוא ריבוע קסם. הנח שהריבוע קטן מ-10*10.

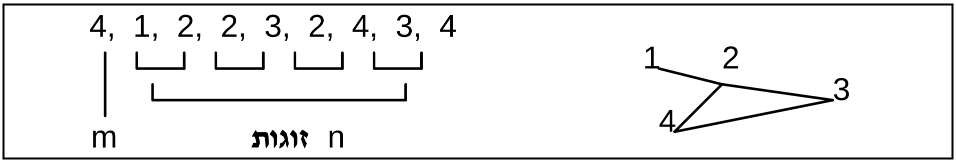
4. באזור מסוים יש רשת כבישים בין ערים. הערים ממוספרות בסדר עולה: 1, 2, ..., 5 וכו'.

יש לכתוב תוכנית שתקבל כנתונים:

m - מספר הכבישים

n - זוגות של מספרים המייצגים ערים, שיש ביניהן כביש ישיר.

לדוגמה, עבור המפה הבאה יהיה הקלט כדלהלן (קרא משמאל לימין):



לאחר שלב זה תדפיס התוכנית הודעה, ותחל לקלוט זוגות מספרים של ערים. לאחר כל קלט של זוג מספרים, התוכנית תבדוק אם הוגדר ביניהן מסלול ותדפיס אותו. אם המסלול לא קיים, תודפס הודעה מתאימה.

דוגמה לפלט:

```
insert pair of cities: 1,2
route between 1 and 2: 1,2
```

(קיים מסלול ישיר)

```
insert pair of cities: 1,4
route between 1 and 4: 1,2,4
```

(קיים מסלול עקיף)

5. יש לכתוב תוכנית המקבלת שני מערכים חד-מימדיים הממוינים בסדר עולה, ואת אורכיהם. התוכנית תדפיס את איברי שתי הסדרות ברשימה אחת, בסדר עולה. איבר המופיע בשני המערכים יודפס רק פעם אחת. אין להשתמש במערך נוסף.

הרץ את התוכנית עבור:

```
-3,1,5,7,11,22,55,72
-3,0,15,22,71,72,96
```

6. נתונה מטריצה בגודל m*n. כתוב תוכנית המכילה 2 פונקציות:

```
switch_col (col1, col2)
switch_row (row1, row2)
```

פונקציות אלו מחליפות 2 עמודות או 2 שורות בהתאמה.

התוכנית תתייחס למטריצה קיימת. היא תחליף בין עמודות כאשר יוקלד התו "c" ואחריו שני מספרים של העמודות והיא תחליף בין שורות כאשר יוקלד התו "r" ואחריו שני מספרים של השורות. כל תו אחר יגרום להדפסת הודעת שגיאה. הקשת התו - "e" תגרום לסיום התוכנית.

7. נתון זוג וקטורים:

```
v0 = (a1,a2,a3,...an)
v1 = (b1,b2,b3,...bn)
```

וקטורים אלה נקראים "אורתוגונליים", אם ורק אם, המכפלה הסקלרית שלהם היא אפס:

```
v0 * v1 = a1 * b1 + a2 * b2 + ... + an * bn = 0
```

קבוצת וקטורים הם אורתוגונליים אם כל זוג בקבוצה הינו אורתוגונלי.

הגדר ואתחל מערך דו-מימדי שיכיל קבוצת וקטורים כלשהי. כתוב תוכנית, אשר תבדוק אם קבוצה זו היא אורתוגונלית.

8. כתוב פונקציה ()str_find המקבלת כקלט שתי מחרוזות str1 ו-str2 ובודקת, אם str1 מוכלת ב-str2. הפונקציה תחזיר 1 אם str1 מוכלת, ו-0 אם לא.

לדוגמה:

```
str_find("st", "string");
  1

str_find("string", "st");
  0
```

9. כתוב פונקציה ()str_n_cmp המקבלת כקלט שתי מחרוזות str1, str2 ומספר שלם n ובודקת, אם n התווים הראשונים בשתי המחרוזות זהים.

הפונקציה תחזיר ערך בדומה ל-()strcmp.

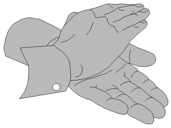
10. כתוב פונקציה ()str_to_up המקבלת מחרוזת והופכת כל אות קטנה לאות גדולה.

דוגמה:

```
"IsraEl" --> "ISRAEL"
```

11. נתון מערך של מחרוזות. המחרוזת הגדולה ביותר היא בת 100 תווים ומספר המחרוזות - 30.

כתוב תוכנית הממיינת מחרוזות לפי סדר לקסיקוגרפי (כלומר, לפי סדר האלפבית) ומדפיסה אותן כשהן ממוינות.



משימות:

תרגילים 6-12.

פרק 7

עיצוב תוכנית

נכתב על ידי אבי בוץ

המונח "**הנדסת תוכנה**" נועד לתאר את תהליך הניתוח של מערכת תוכנה. כאשר ניגשים לפרויקט חדש או מתכננים מוצר חדש, יש לשים דגש על "מה רוצים לבצע", ולא "איך נבצע". יש לתכנן את מהלך הפתרון ולהקדיש לכך מספיק זמן (כאשר המחשב כבוי) מפאת מורכבות התוכניות. בדרך זו נחסוך לעצמנו כאב ראש עתידי מיותר.

מחזור פיתוח תוכנה כולל כמה שלבים עיקריים:

1. הגדרה מפורטת ומדויקת של דרישות המשתמש.
2. עיצוב תוכנית על-פי דרישות אלו.
3. בחירת כלים וקידוד מתאים.
4. בדיקת המערכת.

הגדרת דרישות

כשניגשים לפרויקט, יש לסכם עם הלקוח בכתב את הגדרת דרישותיו. כאשר עומדים לפני פיתוח מוצר חדש לשוק, יש לערוך סקר שוק (או להזמין סקר אצל העוסקים בכך). התוצאה אחת: הגדרת הדרישות ממוצר התוכנה שמפתחים.

רצוי לבנות אב-טיפוס כדי להציגו ללקוח, ולוודא שאכן הלקוח מקבל את מה שהוא רוצה. בונים דגם אב-טיפוס של המערכת בעזרת כלי פיתוח מהירים מהדור הרביעי, שאינם בהכרח כלי הפיתוח של המערכת. לעיתים לא ניתן להגדיר את הדרישות במלואן ובפירוט מספיק ללא כלי עזר זה. בעזרת אב-הטיפוס ניתן לדייק יותר בהגדרת הדרישות לשנות, לבטל, או להוסיף דרישות. זאת עושים הרי כדי להימנע מאי הבנות בעתיד. וזאת אף על פי שאי-אפשר לחזות מראש את כל המצבים ולסגור את כל הפינות. צריך לתכנן את המערכת באופן גמיש, המאפשר שינויים ותוספות, אם וכאשר יידרשו.

עיצוב

עיצוב התוכנית

בשלב זה אנו מתרגמים את הגדרת הדרישות למשבצות ביצוע. כלומר, מה הם המודולים המרכיבים את התוכנית, ומה מרכיב כל מודול. פירוט מרכיבי המודול, כולל את הפונקציות השונות. תיעוד כל פונקציה אשר כולל מה בדיוק אנו רוצים לקבל ממנה. במידת הצורך יש לפרט את האלוגריתמים, או סוגי האלוגריתמים הנדרשים. יש לקחת בחשבון את אילוצי החומרה השונים, תצורות גרפיות, ניצול זיכרון, ועוד. צריך לציין שימוש בשירות ספריות פונקציות קיימות, או תוכנות עזר מכל סוג שהוא, היכולות להשתלב ולעזור ובכך לקצר את זמן הפיתוח.

עיצוב מסכים

עיצוב מסכים כולל חלוקת המסכים על-פי נושאים שונים, עם פירוט של מרכיבי המסך, סגנון עיצוב, צבעים, ועוד. מומלץ להחליט על תקן לעיצוב מסכים ולבנות מסכים על-פי תקן זה.

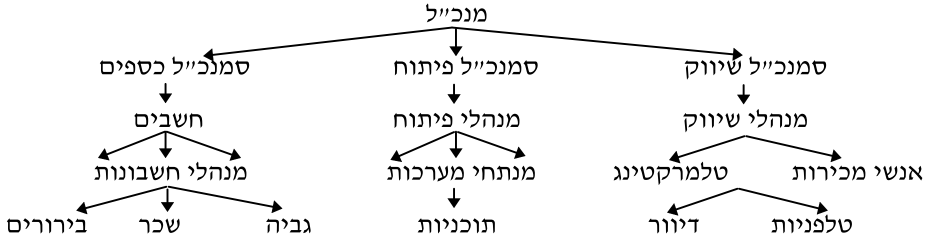
מסכי עבודה צריכים להיות ידידותיים למשתמש, ובמילים אחרות, דרוש ממשק משתמש ידידותי, פשוט ואחיד כדי להקל על המשתמשים. לא פחות חשובה צורת התצוגה - קווים נקיים וברורים. מסכים שאינם עמוסים מוסיפים המון לאיכות המוצר.

מערכת עזרה מקוונת וברורה הכתובה בשפת אנוש ולא בשפת מחשב יכולה לתרום לשימוש נכון ונוח בתוכנית. כלל זה חל גם על הודעות השגיאה למיניהן, כמו בעת קליטת נתון שגוי, או ביצוע פעולה שאינה מותרת.

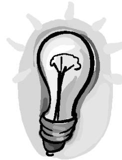
טכניקות עיצוב

טכניקת "Top-Down" - "מעלה-מטה"

בטכניקה זו מבצעים חלוקה של התוכנית לרמות עד לרמה הבסיסית ביותר. ניתן להשוות טכניקה זו לארגון הירככי ראה תרשים 7.1.



תרשים 7.1 : ***טכניקת "מעלה-מטה"***



טיפ!

יש לציין שטכניקת עיצוב זו היא המומלצת, מפני שזו דרך חשיבתו הטבעית של האדם.

להלן תוכנית דוגמה הממחישה את טכניקת העיצוב "Top-Down".

```
               /*    chp0701.c :קובץ תקליטור    */

#include <stdio.h>
#include <conio.h>

                                /* הכרזה על הפונקציות המשתתפות */
int add(int a, int b);
int mult(int a, int b);
void do_what(char action, int a, int b);
void show(int result, int a, int b, char action);

                           /* פונקצית main היא סמנכ"ל הפיתוח */
void main(void)
 {
  int a = 4, b = 5;

  clrscr();                                      /* נקה את המסך */
  do_what('+', a, b);
  do_what('*', a, b);
 }

                           /* פונקציה do_what היא מנהל הפיתוח */
void do_what(char action, int a, int b)
 {
   int result;

   switch (action)
    {
     case '+' :    result = add(a, b);
                   show(result, a, b, action);
                   break;
     case '*' :    result = mult(a, b);
                   show(result, a, b, action);
                   break;
     default:      printf("action error");
    }
 }
```

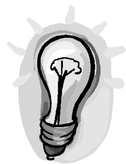
```
                              /* הם התוכניתן add, mult, show פונקציות */
int add(int a, int b)
 {
  return(a + b);
 }

int mult(int a, int b)
 {
  return(a * b);
 }

void show(int result, int a, int b, char action)
 {
  printf("%d %c %d = %d\n", a, action, b, result);
 }
```

טכניקת "Bottom-Up" - "מטה-מעלה"

טכניקה זו היא היפוכה של טכניקת "Top-Down". בטכניקה זו אנו מתחילים ברמה הבסיסית ביותר ומתקדמים אל הרמה העיקרית, הגבוהה.



טיפ!

יש לציין ששילוב בין שתי הטכניקות די מקובל, אך טכניקת "Top-Down" נוחה יותר ברמת תפיסתו הטבעית של האדם.

ראוי לציין, שטכניקת מטה-מעלה גמישה פחות לשינויים, כי לא ניתן לקרוא לפונקציות אשר לא הוגדרו בתחילת התוכנית ומוקמו מתחת לפונקציה הקוראת. זאת אומרת, כאשר ברצוננו להוסיף מרכיב או תת-מרכיב בהירכיה, יש לדאוג למקמו מעל הפונקציה אשר אמורה לקרוא אותו שכן, אחרת הוא לא יוכר על ידי המהדר מהסיבה הפשוטה: הוא פשוט לא ראה אותו קודם. מגבלה זו פוגמת לעיתים ברצף ההגיוני ומכניסה את התוכניתן לאילוצים מיותרים.

לפניך תוכנית דוגמה, הממחישה את טכניקת Bottom-Up:

```
                 /*    chp0702.c :קובץ תקליטור     */
#include <stdio.h>
#include <conio.h>
                                  /* פונקציות show, mult, add, הן התוכניתן */
int add(int a, int b)
 {
  return(a + b);
 }
```

```
int mult(int a, int b)
 {
   return(a * b);
 }

void show(int result, int a, int b, char action)
 {
  printf("%d %c %d = %d\n", a, action, b, result);
 }

                                     /* פונקציה do_what היא מנהל הפיתוח */
void do_what(char action, int a, int b)
 {
   int result;

   switch (action)
    {
     case '+' :    result = add(a, b);
                   show(result, a, b, action);
                   break;
     case '*' :    result = mult(a, b);
                   show(result, a, b, action);
                   break;
     default:      printf("action error");
    }
 }

                                      /* פונקציה main היא סמנכ״ל הפיתוח */
void main(void)
 {
  int a = 4, b = 5;

  clrscr();                                           /* נקה את המסך */
  do_what('+', a, b);
  do_what('*', a, b);
 }
```

סיכום טכניקות עיצוב

שימוש בשיטת עיצוב מודולרית "Bottom-Up" או "Top-Down", מאלץ אותנו להשקיע זמן ומחשבה, אך רק כך אנו יכולים להימנע מראש מחלק מהבעיות אשר יצוצו בהמשך הפיתוח. המערכות כיום מורכבות ביותר, ולא ייתכן שניגש ישר למלאכת התכנות. ללא תכנון מוקדם, אנו פשוט פותחים דלת לבעיות שניתן למנוע אותן מראש.

יש מספר נושאים נוספים שעלינו להתייחס אליהם בשלב זה והם:

ניהול הנתונים. כיצד נאחסן את נתוני המערכת: בקובץ אחד או בסדרת קבצים, מה יהיה מבנה הקבצים? האם נחזיק את הנתונים במסד נתונים טבלאי במקום סדרת קבצים ללא קשר ביניהם. מסד נתונים נפוץ כיום הוא Access של חברת מיקרוסופט.

אבטחת הנתונים. גישה חופשית לנתונים אינה מקובלת, שכן לא ניתן לעובד להציץ במשכורות העובדים אם אינו מורשה לכך, ובוודאי שלא לבצע שינויים שאינו מוסמך לעשותם. יש לקחת בחשבון את נושא אבטחת הנתונים מראש.

בחירת הכלים וקידוד מתאים

מקובל להשתמש בספריות עבודה אשר מסופקות על ידי יצרן המהדר, וגם לרכוש ספריות מיצרנים אחרים. דבר זה חוסך הרבה מאוד זמן, כי משתמשים בחומר קיים ולא ממציאים "את הגלגל". יש לבדוק את איכות חבילת הספריות, לבדוק תיעוד ותמיכה, ולבדוק את העלות. אם בכל זאת הגענו למסקנה שאין מנוס מקידוד עצמי, יש לכתוב פונקציות כלליות ככל האפשר, כדי שניתן יהיה לעשות בהם שימוש חוזר בפרויקטים הבאים.

יש לשים לב לכלים מהדור הרביעי ולבדוק אם ניתן לפתח בעזרתם את הפרויקט, או להפוך אותם לכלי עיקרי ולכתוב ספריות בשפות C/C++ כדי להשלים תכונות אשר חסרות בהם. בשיטה זו, זמני הביצוע מתקצרים ואיכות המערכת עולה. כלים מקובלים אשר נעשה בהם שימוש נרחב הם Access, Visual Basic, Delphi, Power Builder וכד'. יש לקחת בחשבון את אפשרויות הפיתוח בכלים אלה לפני שניגשים לעבודה.

בדיקת המערכת

מערכת טיפול בשגיאות צריכה להבטיח שלא תהיינה "נפילות" של התוכנית, ואם כן - שניתן יהיה לשקם את התוכנית ולהחזירה לתפקוד. מערכת זו גם צריכה להבטיח ששגיאה בתוכנית לא תפגע בתוכניות אחרות שרצות במחשב באותו זמן. בדיקת כל נתון קלט, כל פעולת חישוב הן דוגמאות לנושא זה. במקרה של שגיאה שנובעת מפעולת המשתמש, צריך לתת לו הודעה מפורטת וברורה, כדי שיוכל לתקן או לפעול בצורה נכונה.

יש לבדוק כל פונקציה בנפרד בזמן הפיתוח לפני שמשלבים אותה במערכת, וגם לדאוג לצימוד חלש בין פונקציה לפונקציה. זאת אומרת להשתדל לתכנן את הפונקציות בצורה כזאת שיהוו יחידות עצמאיות עם משתנים מקומיים, ולא להרבות במספר הפרמטרים המועברים אליהן. כך נבטיח פחות שגיאות. רצוי להימנע ככל האפשר ממשתנים גלובליים אשר מהווים פתח לתקלות, בגלל חוסר שליטה.

מומלץ לרכז את הטיפול בשגיאות במודול נפרד. גישה זו תאפשר ניהול אחיד ומסודר של הטיפול בשגיאות.

כתיבה עצמית של פונקציות, תוך שימוש יעיל במערכת הפיתוח של בורלנד, או מיקרוסופט מהווה חיסכון נוסף בזמן. לצורך כך נשתמש בספרות העזר אשר קיבלנו על-מנת ללמוד את רזי מערכת הפיתוח.

לקידוד התוכנית מקדישים בדרך-כלל 40% מזמן הפיתוח, המפתח צריך לפעול לפי נהלי עבודה שנקבעו ביחידה או בארגון. סגנון הכתיבה חשוב אף הוא. כדי להפוך את התוכנית לקריאה יותר לא רק למפתח עצמו, יש לאמץ מערכת כללים לתכנות בשפת C, ולדאוג להטמיעה במחלקות הפיתוח בחברה (שים לב לנספח ב', **נהלים לכתיבת תוכנית ב-C**).

תיעוד

תיעוד יעיל ישפר את התחזוקה של המערכת, מכיון שהמערכת צריכה להתקיים גם לאחר שהמפתח אינו מטפל בה יותר. כאן נעיין בשני היבטים:

שמות משתנים שמות המשתנים, הקבועים, הפונקציות ואפילו שמות הקבצים (*.C) צריכים להיות בעלי משמעות. משמעות זו תתרום רבות להבנת התוכנית. לדוגמה:

```
char get_user_name (char sID_key [ ]);

void main (void)
{
   char sID_key [12];
   char sUser_name [15];

   printf("%s", get_user_name (sID_key));
}

char get_user_name (char sID_key [ ] )
{
   ...

}
```

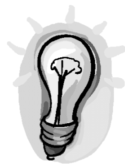
ברור לנו שפונקציה זו מחזירה את שם המשתמש לפי תעודת הזהות שלו.

הערות בגוף התוכנית

הערות מאפשרות להבין טוב יותר את שורות הקוד ומקלות על תחזוקת התוכנית. את הערות נכתוב רק כאשר יהיה צורך בהן. כאשר קטע קוד מורכב יש להסבירו הן לעצמנו, כאשר נחזור אליו לצורך תיקונים או שיפורים והן למפתחים, שיבואו אחרינו. הערות אלו יקלו על חייהם, ויחסכו זמן וכסף לארגון. ניתן לחלק את התוכנית לפסקאות ולתת-נושאים שונים, להסביר מדוע השתמשנו באלוגריתם זה או אחר. מומלץ לא להרבות בהערות, וגם לא לדחות את כתיבת ההערות לסוף הפרויקט, כי לא תמיד יימצא הזמן לכך. חשוב לכתוב את ההערות בעת התכנות.

הערות כותרת

הערת כותרת לתוכנית והערות כותרת לפונקציות חשובות במיוחד. בהערת כותרת לתוכנית עלינו לדעת את שם התוכנית, תאריך ושעת כתיבה, מטרת התוכנית, מי כתב אותה, ולציין הערות מיוחדות אם יש צורך בכך.



טיפ!

מומלץ להכין תבנית הערה, להעבירה מקובץ לקובץ ולשנותה בהתאם.

מומלץ לבנות מפה עם שמות הפונקציות כדי לוודא שאכן יש קשר בין הפונקציות, האם באמת לכל הפונקציות במודול מסוים יש קשר לנושא המודול. ייתכן ויש לנו פונקציות מיותרות. ייתכן והקשר של הפונקציה לנושא מסוים הוא חזק יותר, ועל כן יש להעבירה למודול אחר. אולי יש פרמטרים מיותרים. מפה זו תהיה לנו לעזר רב מאוד - מומלץ להשתמש בה.

הסברים נוספים בנושא ראה בנספח ב׳: **נהלים לכתיבת תוכנית ב-C**.

פרק 8

יצירת פרויקט

נכתב על ידי **אבי בוץ**

עד כה למדנו את טכניקת השימוש בפונקציות ספריה. למדנו שיש לכלול בתוכנית את הספריה הדרושה. למשל, נכתוב <stdio.h> #include לצורך שימוש בפונקציות הנמצאות בספריית קלט/פלט.

מהי פונקציית ספריה?

פונקציה היא תוכנית מחשב אשר כתובה בשפת C או בשפת Assembly, או שילוב של שתיהן. תוכנית מחשב זו נכתבה עבורנו במהדר C של בורלנד או מיקרוסופט (או מהדר אחר), בדרך בה אנו כותבים את תוכניותינו. כך נוצר קובץ C ובו מספר פונקציות בעלות מכנה משותף, כמו טיפול בקלט/פלט, למשל.

בקובץ התוכנית שכתבנו, אשר קרוי גם **קובץ מקור** (source file), אנו כוללים את הפונקציה הראשית main. את הקובץ הזה אנו ממירים לספריה. לקובץ מתוסף קובץ נוסף באותו שם, אך בעל הסיומת h. למשל, stdio.c ו-stdio.h. **בקובץ הכותר** h (header file) נמצאות כל ההכרזות של הפונקציות, בדיוק כמו בשיטת Top-Down. הכללת ספריה זו בתוכנית נועדה לשלב את ההכרזות בקובץ, כדי שנוכל להשתמש בפונקציות שבקובץ stdio.c.

למעשה, מבלי שנדע, עבדנו עד היום עם מספר קובצי מקור שכללו פונקציות שלא נכתבו על ידינו. כיצד, אם כך, נוכל לבנות קובץ פונקציות משלנו, שנוכל להשתמש בו ככל שנמצא לנכון, בלי לכתוב שוב את הפונקציה, או מבלי להעביר אותה מתוכנית לתוכנית?

יצירת תוכנית ממספר קובצי מקור

שפת C מאפשרת לפרוש את כתיבת התוכנית על פני מספר **קובצי מקור** (source files), בעלי סיומת C. כל קובץ כזה יכול להכיל פונקציה אחת או יותר, המוכרות באופן גלובלי זו לזו. קובץ מקור אחד בלבד חייב להכיל את הפונקציה ()main, שממנה מתחילה ריצת התוכנית. לדוגמה, ייתכן המצב המוצג להלן. הקובץ file1.C מכיל:

```
main()
{
 ...
   func1();
   func2();
 ...
}
func1()
 {
 ...
   func2()
 ...
 }
func3()
 }
 ...
 ...
 }
```

הקובץ file2.C מכיל:

```
func2()
{
 ...
}
```

כלומר, למרות העובדה שהפונקציה ()func2 נמצאת בקובץ file2.C, ניתן לקרוא לה מתוך הפונקציות ()main ו-()func1, המוגדרות בקובץ file1.C.

לאחר תהליך ההידור של התוכניות שבקבצים file1.C ו-file2.C, נוצרים קובצי ביניים בעלי השמות file1.obj ו-file2.obj בהתאמה. תהליך ה**קישור** (link) מחבר את שני קובצי הביניים הללו ואת קובצי ספריית המערכת לקובץ ריצה אחד, המקבל סיומת **EXE**. ניתן להריץ אותו על ידי הקלדת שמו, כאשר נמצאים בפיקוח מערכת DOS. למעשה, כאן אנו רואים באופן ברור יותר שהשתמשנו בשיטה זו באופן סמוי. כאשר כתבנו תוכניות C בקובץ מקור אחד, קראנו למעשה לפונקציות של ספריות המערכת מתוך התוכניות. כידוע, פונקציות אלו הוגדרו בקובצי מקור אחרים, ושולבו בשעת ביצוע הקישור.

סוג נוסף של קבצים הינם קובצי ספריה, בעלי סיומת LIB. בכל אחד מקבצים אלה משולבים כמה קובצי obj., והמקשר יודע לשלוף מתוכם רק את הקטעים הנחוצים לתוכנית. בקבצים אלה לא נעסוק כאן, אולם חשוב לדעת שקובצי ספריות המערכת של Borland C למשל, הינם קבצים מהסוג הזה.

מהו פרויקט?

ב-Borland C מקובל לכנות בשם "פרויקט" תוכנית הפרושה על פני מספר קובצי מקור. התהליך ליצירת קובץ exe על ידי הידור וקישור נקרא **עשיית פרויקט** (Project Make).

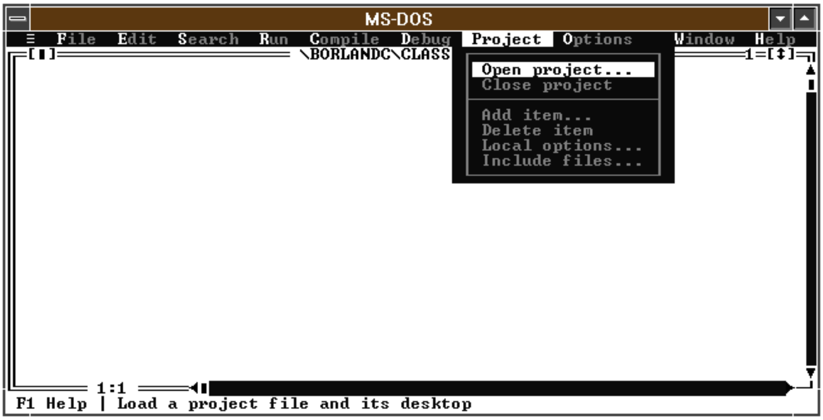
בכדי להגדיר ל-Borland C אילו קובצי מקור שייכים לפרויקט מסוים, מכינים קובץ פרויקט בעל סיומת prj, שבו נמצאת רשימה של קובצי המקור. בדוגמה שלנו, קובץ הפרויקט proj1.prj יכיל את קובצי המקור file1.c ו-file2.c. את קובץ הפרויקט ניתן להכין באמצעות חלון העריכה. לשם כך, יש לבחור את הפריט **P**roject, בתפריט הראשי (או על ידי הקשת Alt+P בכל עת) ולבחור בו את הפריט **P**roject name. כעת יתקבל חלון קלט, שבו ניתן להקיש את שם קובץ הפרויקט: proj1. לאחר מכן, יש לבחור את הפריט **C**ompile מהתפריט הראשי, ובתפריט **C**ompile המתקבל יש לבחור את האופציה **B**uild all. מנקודה זו ואילך, ימשיך המהדר באופן עצמאי. תחילה, הוא יבצע הידור לקבצים file1.C ו-file2.C ויקבל את הקבצים file1.obj ו-file2.obj. אם ההידור מצליח עבור שני הקבצים, יחל תהליך הקישור. קובץ התוצר ייקרא על שם קובץ הפרויקט, proj1.exe אשר ניתן להריץ אותו.

כעת נציג מהלך מפורט של עשיית פרויקט במערכת בורלנד.

כיצד נבנה פרויקט

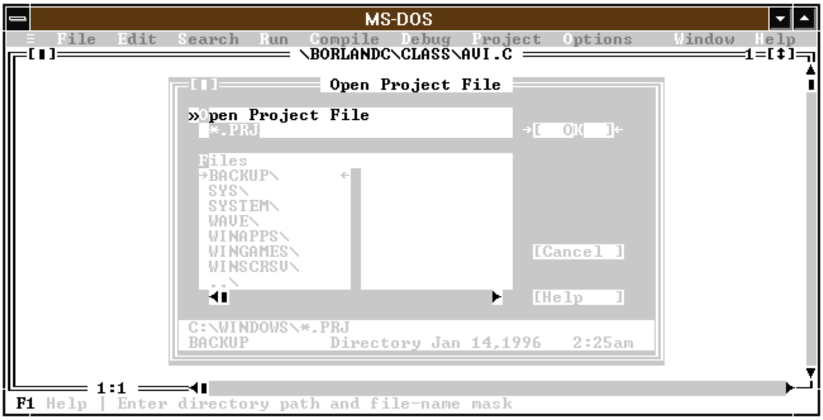
כיצד נבנה פרויקט במערכת Borland C?

1. הצעד הראשון - פתיחת פרויקט חדש.



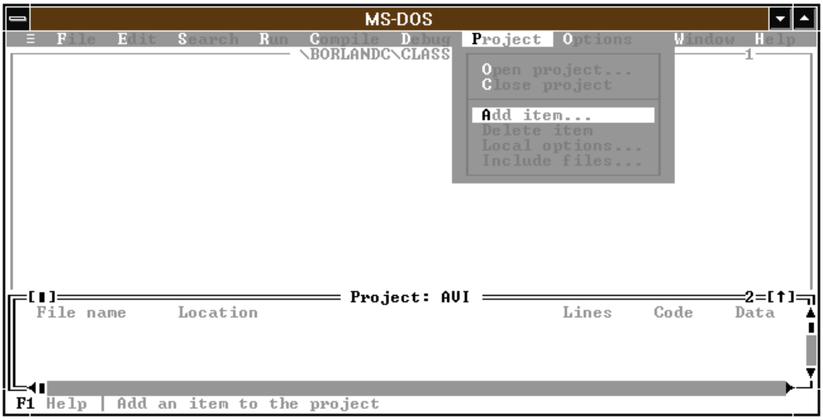
תרשים 8.1: *פתיחת פרויקט חדש*

2. נבחר שם לפרויקט עם סיומת *.prj.



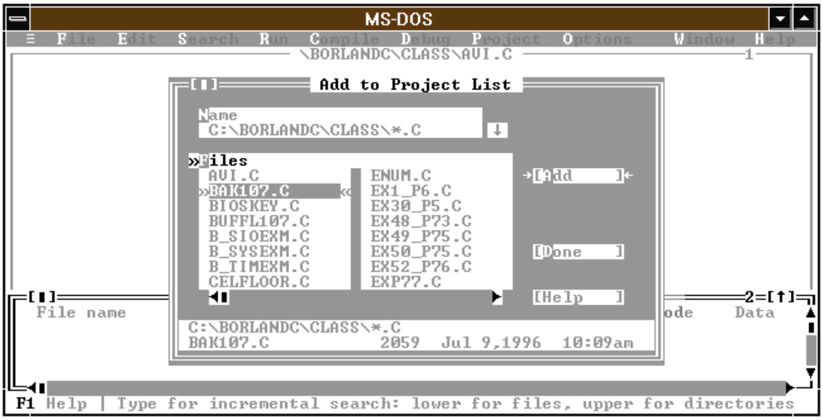
תרשים 8.2: קביעת שם לפרויקט

3. נקבל חלון פרויקט ונכתוב את שמות הקבצים אשר ישתתפו בפרויקט.



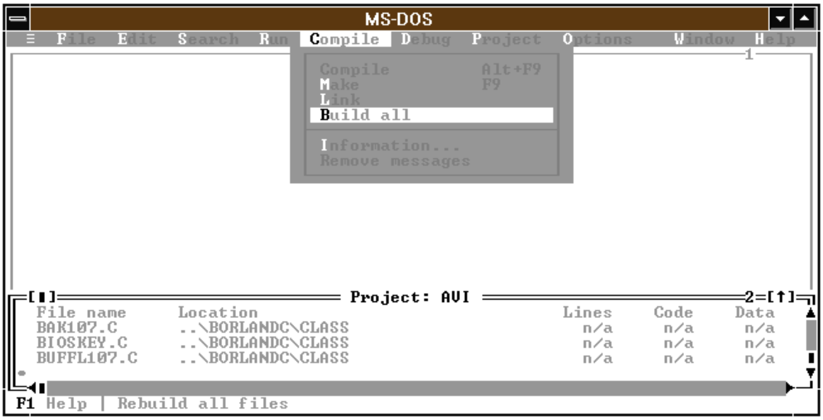
***תרשים 8.3:** נעבור לסימון הקבצים הדרושים*

4. נקבל תיבת דו-שיח. נמקם את הסמן על שם הקובץ ונבחר את הקבצים המשתתפים.



תרשים 8.4: *סימון הקבצים "המשתתפים" בפרויקט*

5. כדי לבנות קובץ ריצה אחד (exe), יש לבחור בפונקציה Build all שבתפריט Compile.



תרשים 8.5: *הוראות ביצוע לבניית הפרויקט*

למעשה, תהליכי ההידור ו/או הקישור לא מבוצעים תמיד. המהדר בודק את התאריך של כל קובץ, ומשווה אותו לתאריך ולשעה של קובץ התוצר, שאמור להיווצר בכל שלב. רק כאשר קובץ המקור **חדש יותר** מקובץ התוצר יבוצע ההידור. לדוגמה, לאחר עשיית הפרויקט נבצע שינוי (בחלון העריכה) לתוכנית file2.C, ומייד לאחר מכן נבצע תהליך עשיית פרויקט. המהדר יבצע את שלב ההידור לקובץ file2.C בלבד, כי על פי התאריך והשעה הוא **חדש יותר** יחסית ל-file2.obj. לא יבוצע כל הידור ל-file1.C, כי הוא לא השתנה מאז ההידור הראשון. לאחר מכן יבוצע שלב הקישור, שכן file2.obj **חדש יותר** מקובץ התוצר proj1.exe. מסיבה זו, חשוב לעדכן את התאריך והשעה במחשב בכל פעם שמתחילים לעבוד (עדכון השעה מתבצע אוטומטית, אם מותקן במחשב כרטיס שעון מגובה סוללה).

מההסבר האחרון ניתן לראות שקיימת מערכת סמיכויות בין הקבצים:

- ❖ file1.obj מסתמך על file1.c.
- ❖ file2.obj מסתמך על file2.c.
- ❖ proj1.exe מסתמך על file1.obj ועל file2.obj.

תוכנית דוגמה לפרויקט

התוכנית הבאה מורכבת משלושה קבצים:

- ❖ קובץ ראשי, mainfile.c המכיל את הפונקציה הראשית main, ומפעיל פונקציות שנמצאות בקובץ file_a.c.
- ❖ קובץ המכיל פונקציות לחיבור ומכפלה של שני מספרים שלמים, וכן פונקציית הזדהות המדווחת על שם הקובץ ומה הוא יודע לעשות.
- ❖ קובץ file_b.c היודע לספר שלוש בדיחות. את הבדיחות בוחרים על-פי מספר הבדיחה. פונקציה זאת מפעילים מהקובץ הראשי mainfile.c.

תהליך חיבור הקבצים ויצירת קובץ הפרויקט הוסבר בסעיף הקודם. התהליך במלואו, על כל שלביו, בוצע על קבצים אלו.

דוגמה:

בקובץ תקליטור chp0801 הקבצים המשתתפים הם:

```
file_a.c
file_a.h
file_b.c
file_b.h
mainfile.c
```

```
        /*****      chp0801.c  :קובץ תקליטור      *****/
/*
File             : mainfile.c
File Type        : Source
קובץ ראשי
*/

#include <stdio.h>
#include <conio.h>
  /*     יש להכליל את ההכרזות על הפונקציות שנמצאות בקבצים אשר בנינו. הכרזות
אלו נועדו לאפשר לנו להשתמש בפונקציות, כדי שהמהדר "יערוך איתם היכרות", שידע
על קיומן. הכרזות אלו מצטרפות להכרזות על פונקציות פנימיות המוגדרות בקובץ זה.
תהליך הבנייה של כל הקבצים לקובץ ריצה אחד מכוון לריכוז כל הפונקציות יחד.   */
```

```
#include "file_a.h"
#include "file_b.h"

   /* הכרזות על פונקציות */
void file_a_action(void);
void file_b_action(void);

/* ----------------------------------------------------------
;  M A I N - P R O G R A M
 ; ------------------------------------------------------*/
int main(void)
 {
   clrscr();
   file_a_action();
   file_b_action();
   printf("Press any key");
   getch();
   return(0);
 }
/* ----------------------------------------------------------
;  F U N C T I O N S
 ; ------------------------------------------------------*/
void file_a_action(void)
 {                  /* file_a.c הפעלת הפונקציות בקובץ */
   file_a_i_am();
   printf("add(int 3, int 4) = %d\n", add(3, 4));
   printf("mult(int 3, int 4) = %d\n", mult(3, 4));
 }

void file_b_action(void)
 {                 /* file_b.c הפעלת הפונקציות בקובץ */
   file_b_i_am();
   joke_selector(1);
   joke_selector(3);
 }
```

```
/*
File            : file_a.c
File Type       : Source
קובץ פונקציות ראשון
*/

#include <stdio.h>
#include <conio.h>
```

```
/* -----------------------------------------------------------
;  F U N C T I O N S
 ; -----------------------------------------------------------*/
void file_a_i_am(void)
 {
  printf("\nשלום! אני קובץ file_a  ואני יודע לחבר ולהכפיל 2 מספרים שלמים\n");
 }

int add(int a, int b)
 {
  return(a + b);
 }

int mult(int a, int b)
 {
  return(a * b);
 }
```

```
/*
File             : file_a.h
File Type        : Header
file_a.c קובץ הכרזות לפונקציות בקובץ
*/

void file_a_i_am(void);
int add(int a, int b);
int mult(int a, int b);
```

```
/*
File             : file_b.c
File Type        : Source
קובץ פונקציות שני
*/
#include <stdio.h>
#include <conio.h>

void file_b_i_am(void)
 {
  printf("\nשלום! אני קובץ file_b  ואני יודע לספר בדיחות\n");
 }
```

```
void joke_selector(int jokeNumber)
 {
  switch (jokeNumber)
   {
     case 1: printf("\nמה אוכלים קניבלים בדיאטה ? גמדים\n");
           break;
     case 2: printf("\n? כמה גרוזינים דרושים בשביל להוריד תמונה מהקיר\n"
                    "עשרה, אחד מחזיק את התמונה ותשעה שוברים ת'קיר\n");
           break;
     case 3: printf("\n? מדוע החבר'ה שחוזרים מגואה לא צריכים אוניברסיטה\n"
                    "כל החומר בראש\n");
           break;
   }
 }
```

```
/*
File            : file_b.h
File Type       : Header
file_b.c קובץ הכרזות לפונקציות בקובץ
*/

void file_b_i_am(void);
void joke_selector(int jokeNumber);
```

תוצאות ריצת התוכנית:

```
שלום! אני קובץ file_a ואני יודע לחבר ולהכפיל 2 מספרים שלמים
add(int 3, int 4) = 7
mult(int 3, int 4) = 12

שלום! אני קובץ file_b ואני יודע לספר בדיחות

מה אוכלים קניבלים בדיאטה? גמדים

מדוע החבר'ה שחוזרים מגואה לא צריכים אוניברסיטה?
כל החומר בראש
```

פרק 9

אפיון משתנים וטווח ההכרה

אפיון משתנים וטווח הכרה

בשפת C, כבשפות עיליות אחרות, **אין חובה** לבצע הידור משותף של כל הפונקציות המרכיבות את תוכנית המחשב. כאשר תוכנית גדולה ומורכבת, מקובל לחלקה למספר חלקים, ולכלול בכל אחד מהם רק חלק מהפונקציות. כל חלק כזה נמצא ב**קובץ מקור** (Source File) וניתן לבצע הידור שלו ללא קשר עם קובצי המקור האחרים. לאחר הידור כל קובצי המקור, ניתן לחבר ביניהם באמצעות תוכנית מיוחדת הנקראת **"מקשר"** (Linker) ולקבל **תוכנית מחשב שלמה המוכנה להרצה במחשב**.

תהליך עבודה זה כבר הוסבר בפרק הקודם. אך עתה נבדוק בכלים של C המאפשרים לנו לעשות כל זאת.

המשתנים המוגדרים בקובצי המקור מסווגים במספר קבוצות, המאופיינות על ידי **טווח ההכרה** (Scope) **ואורך החיים** (Lifetime) שלהן. נגדיר מונחים אלה :

טווח הכרה מגדיר את הפונקציות שבהן יוכר משתנה שהוגדר במקום מסוים, ומאיזה פונקציות ניתן להתייחס אליו.

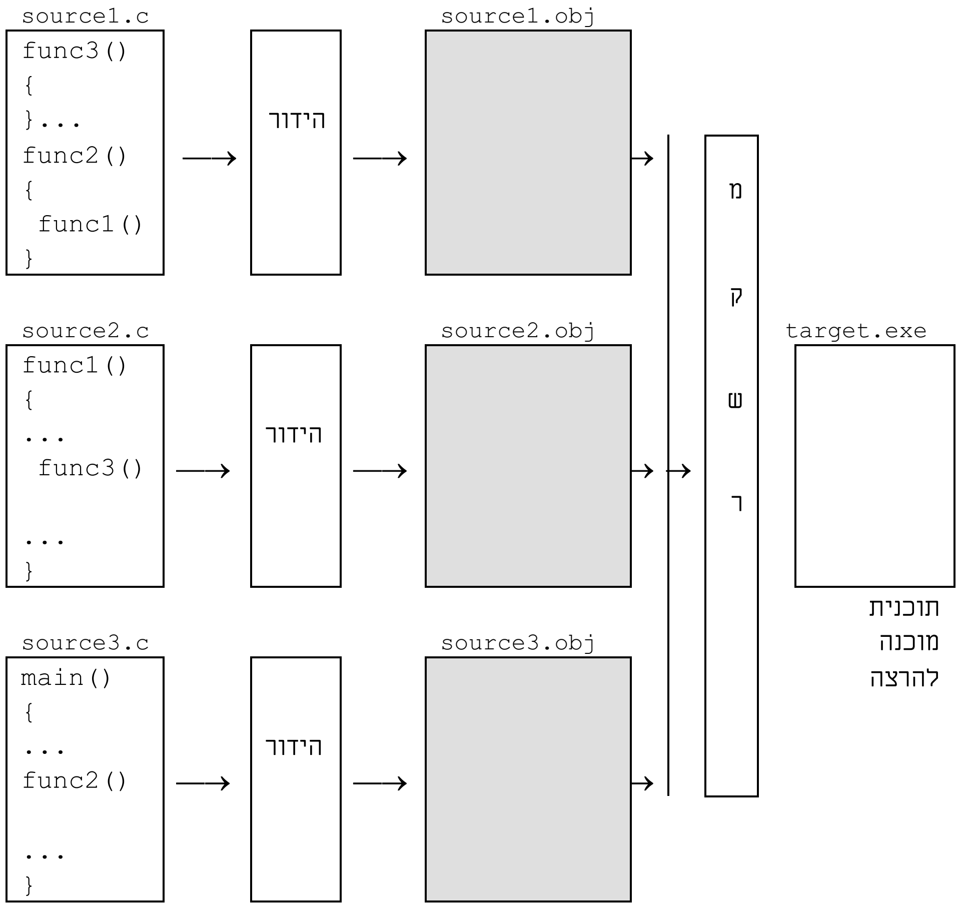
אורך חיים מגדיר באיזה שלב של ביצוע התוכנית נוצר משתנה מסוים, ועד מתי ניתן להתייחס לערכו.

תיאור סכימטי של התהליך ניתן בתרשים *9.1*.

משתנים אוטומטיים

עד עתה עסקנו במשתנים השייכים לקבוצת המשתנים האוטומטיים. משתנים אלה הם הפרמטרים הפורמליים של הפונקציה, או המשתנים המוגדרים בתחילת גוף הפונקציה (מכנים אותם בשם משתנים פנימיים, לוקליים, או משתנים מקומיים). **טווח ההכרה** שלהם הוא בתוך הפונקציה שבה הוגדרו, **ואורך החיים** שלהם הוא משך זמן ביצוע הפונקציה.

מהי המשמעות של עובדה זו? כידוע, כל משתנה תופס מקום מסוים בזיכרון בגודל המתאים לטיפוס שלו. כאשר מפעילים פונקציה, מוקצה באופן אוטומטי מקום בזיכרון עבור המשתנים האוטומטיים שלה, מתוך מאגר הזיכרון הכללי של התוכנית. במהלך ביצוע הפונקציה (וגם כאשר קוראים לפונקציה מתוך פונקציה) נשמרים זהות המשתנים האוטומטיים והערכים שלהם בזיכרון שהוקצה לשם כך. **כאשר הפונקציה מסתיימת, חוזרת הבקרה לתוכנית הקוראת, והזיכרון שהוקצה מוחזר למאגר הכללי**. במקרה זה **אובדים** הערכים של המשתנים האוטומטיים.



***תרשים 9.1:* תהליך ההידור והקישור - יצירת פרויקט**

משתנים כלליים (גלובליים)

קבוצה נוספת של משתנים הם **המשתנים הגלובליים**, אשר **מוגדרים** בתוך קובץ מקור כלשהו **מחוץ** לפונקציות. ניתן להתייחס אליהם בפונקציות אחרות, וערכיהם נשמרים כל זמן ריצת התוכנית, כך מקבלים מנגנון נוסף להעברת ערכים מפונקציה אחת לאחרת. **טווח ההכרה** של המשתנים הגלובליים הוא בכל הפונקציות המוגדרות, החל מנקודת ההגדרה של המשתנים הגלובליים ועד סוף קובץ המקור, **ואורך החיים** שלהם

הוא כל משך ריצת התוכנית. הזיכרון המוקצה עבור המשתנים הגלובליים ישתחרר רק בסיום התוכנית, ולכן כל הערכים נשמרים במשך הריצה שלה.

ניתן להשתמש במשתנים גלובליים גם בקובצי מקור שונים מהקובץ בו הוגדרו. כדי לעשות זאת, על הפונקציה הקוראת להכריז (שים לב, **הכרזה** ולא הגדרה) על המשתנה המסוים שברצונה להשתמש בו כמשתנה גלובלי. הכרזה כזו מציינת למהדר שמשתנה זה מוגדר בקובץ מקור אחר, **חיצוני** (External). צורת ההכרזה:

```
extern <טיפוס> <שם המשתנה>;
```

הכרזה כזו **בתוך** פונקציה טובה עבור אותה הפונקציה, ואילו הכרזה כזו **מחוץ** לפונקציה - טובה עבור כל הפונקציות מאותה נקודה, ועד לסוף קובץ המקור.

נציג תוכנית דוגמה לשימוש באופציית extern לשיתוף משתנים גלובליים בין קובצי מקור שונים.

```
        /*****      chp0901.c :קובץ תקליטור       *****/

/* קובץ ראשי .extern-שימוש ב */

#include <stdio.h>
#include <conio.h>
#include "chp0902.h"

/* הכרזות על פונקציות אשר הוגדרו בקובץ זה */
void extern_m_f1(void);
void extern_m_f2(void);

/* הכרזות extern למשתנים גלובלים, אשר הוגדרו בקובץ מקור אחר (chp0902.c)
   הכרזה זו מציינת למהדר שהמשתנים הוגדרו בקובץ מקור אחר והם למעשה גלובליים
   לשני הקבצים. אפשר לראות זאת כמשתנה גלובלי לקובץ אחד גדול, אשר ניתן לעדכנו
   מכל מקום בתוכנית. מומלץ להימנע משימוש במשתנים גלובליים. */
/* מערך חיצוני יוכר בכל התוכנית */
extern char extern_sName[];
/* משתנה חיצוני יוכר בכל התוכנית */
extern int global_extern_var;

/* -------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ----------------------------------------------------*/
int main(void)
 {
  /* הכרזה על משתנה אשר הוגדר כגלובלי בקובץ המקור cph0902.c משתנה זה
     יוכר בפונקציית main בלבד. לא ניתן לאתחל משתנה חיצוני בשורת ההכרזה. */
   extern int extern_var;
   clrscr();
/* הפעלת הפונקציות המקומיות, אשר הוגדרו בקובץ זה */
```

```
  extern_m_f1();
  extern_m_f2();
  printf("(extern_m->main) extern_var = %d\n",
                                 (extern_var);
  extern_a_f1();       /* chp0902.h הפעלת פונקציה השייכת לקובץ */
/*                     ניתן לשנות את ערכו של משתנה זה. יש לזכור, שכל
                6 קובצי המקור "הרואים" משתנה זה, יקבלו אותו עם הערך */
  extern_var = 6;
  extern_a_f1();       /* chp0902.h הפעלת פונקציה השייכת לקובץ */
                /* .הדפסת ערכי המשתנים הגלובליים תוך ציון מקום ההדפסה */
  printf("(extern_m->main) extern_var = %d\n",extern_var);
  printf("(extern_m->main) global_extern_var = %d\n",
                              global_extern_var);
  printf("(extern_m->main) extern_sName = %s\n",
                                   extern_sName);
  printf("\nPress any key");
  getch();
  return(0);
 }
/* -----------------------------------------------------------
 ;  F U N C T I O N S
 ; ---------------------------------------------------------*/
void extern_m_f1(void)
 {
              /*     .הדפסת ערכי המשתנים הגלובליים תוך ציון מקום ההדפסה */
  printf("(extern_m_f1) global_extern_var = %d\n",
                              global_extern_var);
  printf("(extern_m_f1) extern_sName = %s\n",
                                   extern_sName);
 }

void extern_m_f2(void)
 {
                 /*הדפסת ערכי המשתנים הגלובליים תוך ציון מקום ההדפסה */
  printf("(extern_m_f2) global_extern_var = %d\n",
                              global_extern_var);
  printf("(extern_m_f2) extern_sName = %s\n",
                                   extern_sName);
 }
```

```
        /*****     chp0902.c :קובץ תקליטור      *****/
/* קובץ מקור .extern-שימוש ב */
#include <stdio.h>
#include <conio.h>

/* הגדרת משתנים גלובליים ואתחולם */
int extern_var = 9;
int global_extern_var = 10;
char extern_sName[5] = "Avi";
/* ----------------------------------------------------------
 ;  F U N C T I O N S
 ; -------------------------------------------------------*/
void extern_a_f1(void)
 {
                    /* הדפסת ערכי המשתנים הגלובליים תוך ציון מקום
                                f1 פונקציה chp0902 ההדפסה. קובץ */
  printf("(extern_a_f1) extern_var = %d\n", extern_var);
  printf("(extern_a_f1) global_extern_var = %d\n",
                                          global_extern_var);
  printf("(extern_a_f1) extern_sName = %s\n",
                                               extern_sName);
 }

        /*****     chp0902.h :קובץ תקליטור      *****/
*/
void extern_a_f1(void);
```

תוצאות ריצת התוכנית:

```
(extern_m_f1)global_extern_var = 10
(extern_m_f1)extern_sName = Avi
(extern_m_f2)global_extern_var = 10
(extern_m_f2)extern_sName = Avi
(extern_m->main) extern_var = 9
(extern_a_f1)extern_var = 9
(extern_a_f1)global_extern_var = 10
(extern_a_f1)extern_sName = Avi
(extern_a_f1)extern_var = 6
(extern_a_f1)global_extern_var = 10
(extern_a_f1)extern_sName = Avi
(extern_m->main) extern_var = 6
(extern_m->main) global_extern_var = 10
(extern_m->main) extern_sName = Avi

Press any key
```

שים לב!

מומלץ להימנע משימוש במשתנים גלובליים ככל האפשר, אלא אם אין ברירה, כמו לדוגמה כאשר משתנה מסוים דרוש בהרבה מאוד פונקציות.

זכור!

שימוש רב במשתנים גלובליים מקשה על קריאות התוכנית ועל התחזוקה שלה.

שינוי ערך של משתנה גלובלי, כאשר מספר פונקציות אינן "יודעות" על כך דבר, יוביל לתוצאות שגויות.

נקודת ציון!

אם בכל זאת צריך להשתמש במשתנה גלובלי, מומלץ לקבוע לו קידומת g לפני שמו, כמו int gMenuXY, כדי שנזכור שמדובר במשתנה גלובלי, ונקפיד על השימוש בו.

משתנים סטטיים

לעיתים מעוניינים להקנות למשתנים אוטומטיים **אורך חיים** למשך ביצוע התוכנית כולה (בדומה למשתנים גלובליים), אך לשמר את תכונת הפרטיות שלהם מבחינת טווח ההכרה. כלומר, משתנה כזה, שטווח ההכרה שלו מוגבל רק לפונקציה שבה הוגדר, הוא **משתנה סטטי** (Static). נשתמש במשתנה סטטי, כאשר ברצוננו לשמור על הערך שהתקבל בו בזמן ביצוע הפונקציה, עבור הקריאה הבאה לאותה פונקציה.

משתנה סטטי מוגדר כך:

```
static <טיפוס> <שם המשתנה>;
```

אם ברצוננו שמשתנה מסוים יוכר על ידי מספר פונקציות בקובץ מקור מסוים ואך ורק בהן, נשתמש בהגדרה זו לפני הפונקציה הראשונה (מחוץ להגדרות הפונקציות שבקובץ המקור). משתנה כזה מכונה **סטטי-כללי** (Global Static) ויוכר מנקודת הגדרתו עד סוף קובץ המקור, ואך ורק בקובץ זה. כך מאפשרים לקבוצת פונקציות לשמור על הפרטיות של המשתנים שלהן, ועם זאת - להשתמש בהם במשותף.

הסכימה שבתרשים 9.2 מתארת את הקבוצות השונות וטווחי ההכרה שלהן.

קובץ מקור 1

	glob char ch	glob int a	auto int x	glob int b	auto int ch	auto char x	static float f
char ch;	-x-						
int a;	-x-	-x-					
main()	-x-	-x-					
{	-x-	-x-					
int x;	-x-	-x-	-x-				
...	-x-	-x-	-x-				
}	-x-	-x-					
int b;	-x-	-x-		-x-			
fl(ch)		-x-		-x-			
int ch;		-x-		-x-	-x-		
{		-x-		-x-	-x-		
static float f;		-x-		-x-	-x-		-x-
...		-x-		-x-	-x-		-x-
}		-x-		-x-			

תרשים 9.2: טווחי הגדרה של משתנים בקובץ מקור 1

קובץ מקור 2

	glob char ch	glob int a	auto int x	glob int b	auto int ch	auto char x	static float f
extern int b;				-x-			
f2()				-x-			
{				-x-			
extern char ch;	-x- -x-			-x- -x-			
char x;	-x-			-x-		-x-	
...	-x-			-x-		-x-	
}				-x-			
f3()				-x-			
}				-x-			
extern int a;		-x-		-x-			
...		-x-		-x-			
}				-x-			

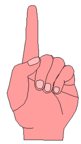
תרשים 9.2 (המשך): טווחי הגדרה של משתנים בקובץ מקור 2

ניתן לראות בטבלאות של הקבצים 1 ו-2, שההגדרה האחרונה של משתנה בעל שם מסוים היא שקובעת, כאשר אזור ההגדרה שונה מהאזור בו הוגדר המשתנה קודם.

במקרה שלנו, המשתנה **ch** מטיפוס תו מוגדר בקובץ מקור 1 כמשתנה גלובלי, וטווח ההכרה שלו הינו עד סוף הקובץ. אולם בפונקציה ()f1, שהיא שטח ריבוני ושונה מאזור ההגדרה המקורי, הוגדר משתנה בעל שם זהה כערך פורמלי, ולכן הוא משתנה אוטומטי מטיפוס שלם. הגדרת ch כמשתנה אוטומטי מטיפוס שלם תופסת לגבי הפונקציה ()f1 בלבד. בפונקציות נוספות בקובץ המקור 1, יוכר שוב המשתנה הגלובלי ch. יש לציין, שניסיון להגדיר משתנה **גלובלי** מטיפוס שלם בשם ch, היה גורם להודעת שגיאה בזמן ההידור, שכן מדובר בהגדרה סותרת **באותו אזור הגדרה**.

המשתנה **b** מוגדר בקובץ מקור 1 כמשתנה גלובלי מעל הפונקציה ()f1, ולכן הוא יוכר מנקודת הגדרתו עד סוף קובץ מקור 1. בקובץ מקור 2, b מוכרז כמשתנה גלובלי מחוץ לכל פונקציה, ולכן הוא יוכר כמשתנה גלובלי מנקודות ההכרזה ועד לסוף קובץ מקור 2. המשתנה **a** מוגדר כגלובלי בקובץ מקור 1, בדומה למשתנה b. אולם בקובץ מקור 2 הוא מוכרז כחיצוני רק בתוך הפונקציה ()f3, ולכן הוא יוכר כגלובלי בתוך פונקציה זו בלבד.

אפשרות נוספת לשימוש באופציית static, היא **הסתרת פונקציה** מפני קבצים אחרים.



שים לב!

כאשר נגדיר פונקציה כ-static, לא ניתן יהיה להפעילה מתוך קובץ מקור אחר. בפעולה זו אנו מסתירים אותה מפני קבצים אחרים.

מדוע זה טוב?

כאשר נחליט שהפונקציה בקובץ המקור הינה פנימית ונועדה לשימוש אך ורק בקובץ המקור על ידי פונקציות אחרות באותו קובץ, למשל. הוספת ההגדרה static לפני הגדרה והכרזה של הפונקציה, תמנע מאתנו להפעילה בטעות מקובץ מקור אחר ובמיוחד - על ידי התוכניתן אחר הנדרש להוסיף את חלקו למערכת, או לתחזק אותה.

בתוכנית הדוגמה הבאה נדגים את השימוש באופציה static להסתרת פונקציות.

```
        /*****       chp0903.c :קובץ תקליטור       *****/

/* להסתרת פונקציות static שימוש באופציה */

#include <stdio.h>
#include <conio.h>
#include "chp0905.h"

/* -----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ---------------------------------------------------------*/
int main(void)
 {
  clrscr();
```

```
  /* לא ניתן להפעיל פונקציה זו מקובץ זה */
  //static_a_f1(); // Error!
  printf("\nPress any key");
  getch();
  return(0);
 }
```

```
        /*****      chp0904.c :קובץ תקליטור       *****/

/* שימוש באופציה static להסתרת פונקציות */

#include <stdio.h>

/* הגדרת פונקציה כ static תמנע מאתנו להפעילה בקובצי מקור
   אחרים, אך ורק בקובץ זה. */
static void static_a_f1(void)
 {
  printf("I am the static function static_a_f1\n");
 }
```

קובץ הכרזות לקובץ static_a.c

```
        /*****      chp0905.c :קובץ תקליטור       *****/

/* שימוש באופציה static להסתרת פונקציות */

/* הכרזה על פונקציה static */
static void static_a_f1(void);
```

משתני אוגר

הקבוצה הרביעית והאחרונה של משתנים בשפת C היא **משתני האוגרים** (Register Variables). קבוצה זו תקפה אך ורק לגבי **משתנים אוטומטיים**. הגדרה של משתנה אוגר היא "המלצה" למהדר ליצור פקודות מכונה מתאימות לשמירת משתני האוגר בתוך האוגרים של היע"מ (יחידת העיבוד המרכזית). זוהי המלצה בלבד, מכיון שמספר האוגרים ביע"מ קטן, ולא תמיד יכול המהדר להקצות אוגר למטרה זו. במקרה שהאוגרים תפוסים, יהיו המשתנים הללו אוטומטיים.

צורת ההגדרה של משתנה אוגר:

```
register <טיפוס> <שם המשתנה>
```

מומלץ להשתמש בהגדרה זו עבור משתנים אוטומטיים שהשימוש בהם שכיח מאוד.

כדוגמה לשימוש במשתני אוגר נציג מוני לולאות:

```
sum( int n)
{
register int i, s;
   for(i = 1, s = 0; i <= n; ++i)
              s += i;
return(s);
}
```

הפונקציה ()sum מחשבת ומחזירה את סכום המספרים מ-1 עד n, שהוא הערך הפורמלי שלה. i ו-s הם משתנים אוטומטיים המשמשים לצורך חישוב הסכום. הגדרתם כמשתני אוגר תיצור (אם הדבר אכן אפשרי) תוכנית יעילה יותר, שכן הפעולות בין האוגרים ביע"מ הן המהירות והחסכוניות ביותר.

נסכם את התכונות של משתני אוגר:

א. מסיבות של גודל וטיפוסי המשתנים - משתני אוגר יכולים להיות מטיפוס תו, שלם, או מצביע בלבד (char, int, *).

ב. מספר המשתנים השונים המוגדרים כמשתני אוגר מוגבל, ותלוי בסוג המחשב בו אנו משתמשים.

ג. הגדרת משתנה אוגר מהווה המלצה בלבד למהדר.

דוגמה לנושא טווחי הכרה

```
        /*****       chp0906.c :קובץ תקליטור        *****/

#include <stdio.h>

void f1(int x);
void f2(int y);
int  x=20, y;

void main(void)
{
   y = 5;
   f1(x);
   f2(y);
   printf("main():(x,y)=(%d,%d)\n",x,y);
}

void f1(int x)
{
   printf("f1():(x,y)=(%d,%d)\n",x,y);
   x = y = 10;
}
```

```
void f2(int y)
{
   printf("f2():(x,y)=(%d,%d)\n",x,y);
   x = y = 15;
}
```

לאחר הרצת התוכנית יתקבל הפלט הבא:

```
f1() : (x,y) =   (20,5)
f2() : (x,y) =   (20,10)
main() : (x,y)   = (15,10)
```

השתמש בסכימת טווחי ההכרה שהוצגה קודם ובכללים שלמדנו עד כה, כדי לאמת את נכונות התוצאות שקיבלנו.

אתחול משתנים

אתחול (Initialization) הינו מתן ערכים התחלתיים למשתנים. פקודת האתחול משולבת בפקודת ההגדרה של המשתנה באופן הבא:

<טיפוס> <שם משתנה> **=** <ביטוי אריתמטי של ערך ראשוני> ;

דוגמה:

```
f1()
{
   int   i=5, j=10;
   char c = 'A' ;
}
```

את קטע התוכנית הזה, אפשר לכתוב בדרך שונה:

```
f1()
{
   int   i, j;
   char c;
   i = 5;
   j = 10;
   c = 'A';
}
```

אתחול של משתנה **גלובלי** או **סטטי** הינו למעשה מתן ערך התחלתי למשתנה בתחילת הריצה של התוכנית. מכיון שמשתנים אלה קיימים עד סוף ביצוע התוכנית, לא תהיה לפקודת האתחול כל השפעה, מעבר לעובדה זו. הערך הראשוני מחושב בזמן ההידור, ולכן הוא יכול להיות ביטוי המורכב מקבועים בלבד!

אתחול של משתנה **אוטומטי** מתבצע בכל פעם שמשתנה זה נוצר כאשר מפעילים את הפונקציה שבה הוא מוגדר. הערך הראשוני מחושב תוך כדי ריצת התוכנית, ולכן הוא

יכול להיות ביטוי המורכב מקבועים ומשתנים. בדוגמה הבאה נשכלל את הפונקציה sum()‎ וננצל את אפשרויות האתחול.

```
(1)  sum(int n)
(2)  {
(3)      register int  i=n, s=0;

(4)      while (i)
(5)       s += i-- ;
(6)      return(s);
(7)  }
```

הסבר:

א. משפט for בדוגמה של משתני אוגר הוחלף (בשורה 4) במשפט while, מכיון שאיננו זקוקים לחלק המאתחל של משפט זה.

ב. הקטנת ערך המונה (i) משולב בשורה 5 בתוך פקודת חיבור מקוצרת, בשיטת הקטנה עצמית מאוחרת.

ג. בשורה 3 מתבצע האתחול של המונה, ואנו מסתמכים כאן על העובדה שמשתנה אוטומטי מאותחל בכל פעם שמופעלת הפונקציה בה הוא מוגדר.

ד. מטעמי נוחות ביצענו בפונקציה הנוכחית חיבור הפוך (מהערך הגבוה אל הערך הנמוך). תוצאת החישוב איננה משתנה כמובן.

כאשר לא מתבצע אתחול מפורש, תקפים הכללים הבאים:

א. משתנים גלובליים וסטטיים מאותחלים לאפס על ידי המהדר.

ב. משתנים אוטומטיים יכילו ערך בלתי מוגדר על פי התכולה האחרונה של המקום שהוקצה עבורם בזיכרון. ערך זה נקרא בלשון עממית "זבל".

פרק 10

הגדרות קדם-מעבד

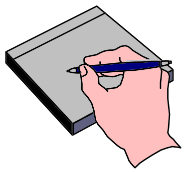
המהדר לשפת C כולל בתוכו **קדם-מעבד** (pre-processor), שתפקידו לתרגם **משפטי בקרה** (Compiler Directives) ולבצע לפיהם פעולות מסוימות בזמן ההידור. ניתן לסווג את משפטי הבקרה לשלוש קבוצות:

File Inclusion הכללת קבצים לפי שם בזמן פעולת ההידור.

Macro Substitution עיבוד והחלפה של הגדרות מאקרו.

Conditional Compilation הידור מותנה.

באמצעות משפטי בקרה אלה, אפשר לייעל את תחזוקת הקבצים, לקצר את גודל ה**תוכנית המורצת** (Executing program) וליצור מכנה משותף למספר רב של קבצים.



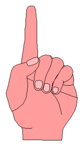
הערה!

טווח ההכרה *(Scope) של משפט בקרה, הוא ממקום ההגדרה ועד סוף קובץ המקור. כפי שנראה בהמשך, ניתן לבטל את השפעת משפט בקרה מנקודה מסוימת בקובץ ואילך.*

במהלך ההידור של תוכנית C יש ארבעה שלבים:

1. מעבר (pre-processor) על קובץ המקור לפני תחילת ההידור. כלומר, תהליך הקודם להידור.

   הפקודות המיוחסות לתהליך pre-processor מאופיינות על ידי ציון "#" (סולמית, דיאז) בתחילת השורה.



שים לב!

תהליך זה גורם בדרך כלל להגדלת קובץ המקור. תהליך #include למשל, מעתיק תוכן קובץ ***כותר*** *(header) לקובץ המקור. התניות הידור קובעות מה ייכנס לקובץ המקור ובאיזה תנאי.*

2. בדיקת תקינות הפקודות מבחינה תחבירית.
3. יצירת קובץ, או קובצי object.
4. קישור בין קובצי object כדי ליצור קובץ EXE (קובץ הרצה).

הכללת קבצים

בדוגמאות השונות שהבאנו במהלך הספר, כללנו פקודה הגורמת לכך שבמקום בו היא נמצאת מוכנס לצורך ההידור תוכן הקובץ stdio.h.

```
#include <stdio.h>
```

הוראת include# מאפשרת להכליל קובץ כלשהו במסגרת קבצים אחרים. שימוש עיקרי לכך הוא בקובצי ההגדרות שאינם מכילים פקודות ממש, אלא הגדרת טיפוסים, קבועים וטיפוסי פונקציות. נהוג לכנות קבצים אלה בשם **קובצי כותר** (header files), או include files, ולתת להם סיומת : h. או inc בהתאם.
ההגדרה מתבצעת בשני אופנים :

```
#include "שם קובץ.h"
#include <שם קובץ.h>
```

במקרה הראשון, יחפש קדם-מעבד את הקובץ בספריה הנוכחית או במסלול אשר נציג לפניו. לדוגמה :

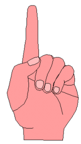
```
#include "c:\borland\book\שם קובץ.h"
```

במקרה השני יתבצע החיפוש במסלול הסטנדרטי :

```
"C:\BORLANDC\INCLUDE"
```

ראוי לציין, כי תוכנם של קובצי header אינו מוגבל, וניתן לכלול בהם תוכניות ופונקציות, אם כי הכללה זו אינה רצויה ואינה מקובלת.

ניתן לבצע קינון של הגדרות אלו, דהיינו הגדרת include# בתוך קובץ שגם הוא נכלל בקובץ אחר.



שים לב!

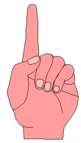
הפקודה include# תגרום להכללת קובץ header בקובץ המקור בזמן ההידור.

עיבוד והחלפה של הגדרות מאקרו

נניח שבתוכנית אנו מתייחסים לקבוע מסוים מספר פעמים במקומות שונים, לדוגמה : אורך מקסימלי של מחרוזת, גודל מערך, ערך מוחזר וכו'. אחרי שהרצנו את התוכנית מספר פעמים הסתבר, שעלינו לשנות בה קבוע מסוים. ללא מנגנון ההגדרה היה עלינו לערוך את קובץ המקור ולשנות את כל המשפטים שבהם מופיע קבוע זה. כאשר תוכנית המקור גדולה במיוחד - פעולת שינוי כזו קשה, ולא אמינה. לעיתים קרובות צריך לבצעה מספר פעמים, כי לא מאתרים ומתקנים בפעם הראשונה את כל המשפטים הרלוונטיים.

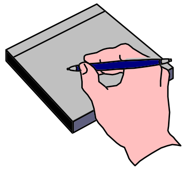
מנגנון ההגדרה מאפשר להחליף את הקבועים בשמות סמליים, ובעזרת שינוי יחיד לבצע את הפעולה שתיארנו קודם. ההגדרה נעשית בצורה הבאה :

```
#define <שם סמלי> <ערך מוחלף>
```



שים לב!

*שם סמלי נקרא גם **קבוע**.*



הערה!

נהוג בשפת C להגדיר שמות סמליים באותיות אנגליות גדולות (Capitals), כדי להבדילם ממשתנים ומילים שמורות.

התוכנית הבאה מדגימה את יעילות מנגנון ההגדרה:

```
         /*****      chp1001.c :קובץ תקליטור        *****/
     #include <stdio.h>                    /*  הדגמת מנגנון הגדרה  */
(1)  #define MAXLEN 20
(2)  #define TRUE 1
(3)  #define FALSE 0
(4)  #define POSITIVE 'y'
(5)  #define NEGATIVE 'n'
(6)       /* the function */
(7)  int fill_array(void)
(8)  {
(9)  int numbers[MAXLEN], count;
(10) char answer;
(11) count = 0;
(12) answer = NEGATIVE;
(13) while ((count < MAXLEN) && (answer == NEGATIVE))
(14)    {
(15)    printf ("\n Enter a number to array,
(16)                   %d remainig places:",MAXLEN-count);
(17)    scanf ("%d", &numbers[count++]);
       fflush(stdin);
(18)    printf ("\n Do you want to finish? <y/n>:");
(19)    scanf ("%c", &answer);
(20)    }
(21) if (count == MAXLEN)
(22)    return(TRUE);
(23) return(FALSE);
(24) }
(25)      /* the program */
(26) void main(void)
(27) {
(28)    char ret_val;
(29)    if ((ret_val = fill_array()) == TRUE)
(30)      printf ("\n Array was filled completely");
```

```
(31)      else
(32)        printf ("\n Array is not full");
(33)   }
```

הסבר:

א. הערך המספרי הקבוע 20 משמש אותנו בארבעה מקומות שונים במהלך התוכנית הקצרה. בעזרת מנגנון ההגדרה (שורה 1) תרמנו ל**קריאות** (readability) של התוכנית ולתחזוקתה. אם נרצה, בעתיד, לשנות את אורכו המקסימלי של המערך number, נוכל להסתפק בשינוי היחיד של ההגדרה:

```
#define MAXLEN <ערך חדש>
```

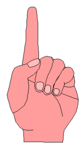
ב. באמצעות המאקרו #define אפשר לבצע פעולות נוספות, כמו למשל לקבל ערכים ולהציבם בביטוי מורכב. כאשר המהדר מגיע להגדרת המאקרו מוחלף השם הסמלי בביטוי כולו. ההגדרה:

```
#define <שם סמלי>(1 פרמטר,...,n פרמטר) <ערך מוחלף>
```

דוגמה:

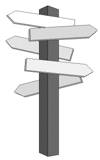
```
#define SUM_DIV_MUL(a,b) a+b/a*b
#define tripple_exp(x) x*x*x
```

לדוגמה, כאשר הקדם-מעבד מגיע לביטוי המכיל tripple_exp(10), הוא יחליף אותו בביטוי 10*10*10. באופן דומה, הביטוי tripple_exp((a+5)) יוחלף ב-(a+5)*(a+5)*(a+5).



שים לב!

הסוגר השמאלי "(" צמוד לשם הסמלי כדי למנוע אי-הבנות בזמן הפענוח של קדם-המעבד.



נקודת ציון!

*השימוש בהגדרת מאקרו באופן זה אינו רצוי בתוכניות שבהן חשוב לקבל **תוכנית ריצה** (Executing) קצרה. השימוש בפונקציה במקום במאקרו מאפשר לחסוך במקום, כי כאשר מציינים בתוך התוכנית שם של פונקציה, מבוצע דילוג אל מקום הפונקציה הנמצאת רק פעם אחת בתוכנית, ולא מעתיקים אותה בכל מקום שהיא נדרשת. מאידך גיסא, השימוש במאקרו יגרום לתוכנית להיות מהירה יותר. רצוי להקיף בסוגריים כל פרמטר בהגדרת המאקרו, כדי למנוע שיבושים כתוצאה מקדימויות.*

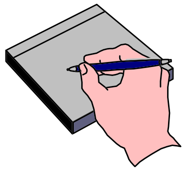
#define יכולה להופיע בכל מקום בקובץ אחר.

בהגדרת #define ניתן לחרוג מגבולות שורה על ידי תו המשך "\". דוגמה:

```
#define MSG printf("Usage : start [File Name.[c].[cpp].[h]\
 [First Name] [Last Name]\n");
```

בכל פעם שתופיע ההודעה MSG בקובץ המקור, היא תוחלף בשורת printf, ובזמן הריצה תוצג ההודעה הכתובה בין הגרשיים.

שימוש נוסף בהוראת ההגדרה מאפשר להימנע ממשפטים ארוכים ומסורבלים.



הערה!

ניתן לבטל הגדרת מאקרו החל ממקום מסוים ואילך באמצעות הפקודה #undef. היא תגרום לכך שהשם הסמלי לא יוכר אחריה (אלא אם הוגדר שנית בעזרת #define). ההגדרה:

```
#undef <שם סמלי>
```

בעזרת מנגנון ההגדרה והמנגנון המותנה שנלמד מייד, ניתן להגדיר יחסי גומלין בין קבצים שונים הקשורים זה לזה.

הידור מותנה

הידור מותנה מאפשר לנו לכלול או להוציא קטעים מסוימים מתוכנית בהתאם לתנאי, או תנאים מסוימים.

א. הקטע יוכלל אם הביטוי אינו 0

```
#if <ביטוי המורכב מקבועים>
```

ב. הקטע יוכלל אם השם מוגדר

```
#ifdef <שם סמלי>
```

ג. הקטע יוכלל אם השם אינו מוגדר

```
#ifndef <שם סמלי>
```

כל הגדרה #ifdef מסיימים במילה #endif, וניתן לכלול ביניהן את המילה #else. ניתן לבדוק מפתחות שונים (אלה אותם שמות סמליים אשר השתמשנו בהם להגדרת תבניות הפקודות), אשר נגדיר תחילה בקובץ המקור. בהתאם לכך יבוצע הידור סלקטיבי של הקטעים השונים של התוכנית.

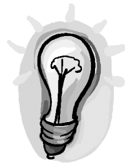
דוגמה לכך יכול להיות קטע שמטפל בטיפוסי קלט שונים ומבצע פעולות אריתמטיות בהתאם:

```
#ifdef  FLOAT_KEY
        float  number1, number2;
#else
        #ifdef  INT_KEY
         int  number1, number2;
        #else
         #ifdef  CHR_KEY
                 char  number1, number2;
         #endif
        #endif
#endif

number1 += number2;
```

הסבר:

אם FLOAT_KEY הוגדר קודם בפקודה define#, תתבצע הפקודה עד ל-else#. אחרת, יתבצעו הפקודות שנמצאות לאחר else# עד ל-define#.



טיפ!

ניתן באותה שיטה לבצע הכללות מותנות של קובצי header.

הבחירה בין הקטעים השונים תהיה באמצעות סדרת מפתחות כדוגמת:

```
#define FLOAT_KEY
#define INT_KEY
#define CHR_KEY
```

הביטוי הנבדק יכול להיות ביטוי מורכב הכולל פעולות אריתמטיות, אבל אך ורק בין קבועים. את ההתניות ניתן לכתוב בקינון, כפי שמוצג בדוגמה.

הגדרות אלו מאפשרות לבדוק אם שמות מסוימים מוגדרים, ולכלול קטעים שונים בהתאם לדוגמה:

```
#ifdef FLOAT_KEY
        #undef FLOAT_KEY
        #define FLOAT_KEY 'F'
#endif
```

או:

```
#ifndef FLOAT_KEY
        #define FLOAT_KEY
#endif
```

בעזרת הגדרות אלו ניתן לקשור בין קבצים רבים, גם אם תוכן חלק מהם אינו ידוע, ולמנוע הודעות שגיאה על הגדרות כפולות, או משתנים שאינם מוגדרים.

דוגמה נוספת להידור מותנה:

במהלך הניפוי של תוכנית ניתן לשבץ הדפסות ביקורת במקומות שונים, ולבטל את הביצוע שלהן כאשר התוכנית מוכנה להרצה.

```
/*****      chp1002.c :קובץ תקליטור       *****/

#include <stdio.h>
                                        /* דוגמה להידור מותנה */
#define DEBUG
void main(void)
{
  int  x, y ;
  for (x = 0; x < 100; ++x)
    {
    if (x < 50 && x > 30)
       y = x;
    if (x > 60)
       y += x;
       #ifdef DEBUG
       printf ("x, y : %d %d \n", x, y);
    #endif
    }
printf ("Result is : %d \n", y);
}
```

בתוכנית זו נקבל את כל ערכי הביניים של x ו-y. לאחר שנוודא שהתוכנית פועלת כראוי, נוכל לבטל את השורה #define DEBUG על ידי מחיקתה, או הכנסתה בתוך הערה, ונבצע הידור. כאשר נריץ את התוכנית, לא תבוצענה ההדפסות של ערכי הביניים.

בתוכנית גדולה ניתן לשבץ הדפסות במקומות שונים לשם ביקורת, ולשלוט על כולן באמצעות שורת ההגדרה של DEBUG. כאשר מכניסים שינויים בתוכנית, ניתן לחדש את פקודת #define DEBUG ולהפעילה שוב עם הדפסות הביקורת בעת הניפוי.

פרק 11

מצביעים

המצביע (Pointer) **הוא מצביע לכתובת בזיכרון**. זהו משתנה בעל גודל קבוע, אשר תלוי בסוג המחשב וגודל מרחב הזיכרון שלו. ניתן לראות במצביע מעין "נקודת ייחוס", אשר בפעולות חשבוניות פשוטות אפשר להזיזו על פני טווחים גדולים של זיכרון, "לדלות" ממנו מידע ולהציב לתוכו מידע אחר. בכך ניתן לייעל את התוכניות הנכתבות.

פונקציות בשפת C מקבלות פרמטרים "**לפי ערך**" - "By value", אשר הדרך היחידה לשנותם הינה על ידי שימוש במצביעים. הגמישות הרבה של השימוש במצביעים בשפת C הופכים אותם לכלי חשוב בשפה זו, אשר מחייב תשומת לב רבה כדי להימנע מתקלות ושגיאות.

עלינו לציין, שלמרות שמצביע הוא משתנה בעל גודל קבוע, הרי כל אחד מסוגי המצביעים מיוחס אל **הטיפוס שעליו הוא מצביע**. התעלמות מעובדה זו היא שגיאה נפוצה בתוכניות הנכתבות בשפת C.

אנו ממליצים לקרוא פרק זה בעיון רב, מכיון שהבנתו תקנה לך כלי חשוב ורב עוצמה לכתיבת תוכניות בעתיד.

מצביעים וכתובות זיכרון

ה**מצביע מצביע אל כתובת בזיכרון**. אם x הוא משתנה מסוג שלם וכתובתו 2000, הרי שהמצביע px המצביע עליו, יכיל כתובת זו. בשפת C משייכים לכל משתנה שני ערכים: **lvalue** ו-**rvalue**, או **ערך שמאלי** (Left value) **וערך ימני** (Right value) בהתאמה. הערך השמאלי הוא כתובת המשתנה בזיכרון והערך הימני הוא הערך שמכיל המשתנה. המצביע מכיל את הערך השמאלי של המשתנה שאליו הוא מצביע. המצביע יכול להיות מוגדר לפני המשתנה, או אחריו.

לדוגמה, בתרשים 11.1 מוצג בצורה סכימטית משתנה x שנמצא בכתובת 2000 וערכו הוא 25.

```
        x
lvalue / \ rvalue

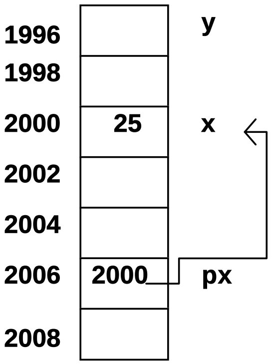
     /      \

  2000     25
```

תרשים 11.1:* *שני ערכים משויכים למשתנה x

דוגמה נוספת:

בהנחה ש-px נמצא בכתובת 2006 בזיכרון, נראה בתרשים 11.2 את תמונת הזיכרון והערכים של x ו-px.




```
      px                     x
lvalue / \ rvalue   lvalue / \ rvalue
      /   \                /   \
2006        ---> 2000 <---      25
```

תרשים 11.2:* *כיצד פועל המצביע

ניתן לראות, שאם נציב ב-px את הערך 1996 הוא יצביע על המשתנה y. נכתוב תוכנית שמתאימה למצב זה.

```
(1)    void main(void)
(2)    {
(3)    int x, y, *px ;
(4)    px = &x;
(5)    *px = 25;
(6)    y = *px;
(7)    }
```

הסבר:

א. לפנינו שני סימנים חדשים: & ו-*. הסימן * (כוכבית) פירושו "**הערך המוצבע על ידי**". לכן, משמעות ההגדרה שבשורה 3 היא: "הערך המוצבע על ידי px הוא מטיפוס שלם". כדי להכריז למשל, שהערך המוצבע על ידי f_ptr הוא מטיפוס ממשי, יש לכתוב:

```
float *f_ptr;
```

ב. הסימן & פירושו "**כתובתו של**". בהתאם לכך מתקיימים השוויונים הבאים:

```
&x = 2000
&y = 1996
```

ומהו הערך שמייצג הביטוי &px ? - ערך זה הינו 2006.

ג. בשורות 4-6 מתבצעות שתי הצבות למשתנים מטיפוס שלם:

```
x = 25
y = x
```

בדרך זו משתמשים במצביע px כדי להציב ערך ל-x. משמעות המשפט y = *px; בשורה 6 היא: "y מקבל את הערך המוצבע על ידי px". כאן, זהו ערכו של x.

מכיון שלכל מצביע משייכים טיפוס, עלינו להצהיר על כך בזמן הגדרתו:

```
char *ptr;
float *f_ptr;
long *l_point;
```

עתה נערוך סימולציה של התוכנית הקודמת להמחשת תהליך העבודה עם מצביעים.

```
/*****      chp1101.c :קובץ תקליטור       *****/

/*  סימולציה של מצביעים למשתנים שלמים  */

#include <stdio.h>
#include <conio.h>

/* ----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; -------------------------------------------------------*/
int main(void)
 {
   int x, y, *px;

   clrscr();
/*  נדפיס את ערכי המשתנים שהוגדרו ונראה שערכם אקראי לחלוטין  */
   printf("x = %d , y = %d , *px = %d\n", x, y, *px);
/*  נדפיס את כתובת המשתנים שהוגדרו בייצוג הקסה-דצימלי  */
   printf("&x = %x , &y = %x , &px = %x\n", &x, &y, &px);
/*  נדפיס את כתובת המשתנים שהוגדרו בייצוג של מצביע  */
   printf("&x = %p , &y = %p , &px = %p\n", &x, &y, &px);
/*  נבדוק לאיזו כתובת המצביע px מצביע ומי מתגורר בכתובת זו,
  ניתן לראות שהמצביע מצביע לעבר כתובת אקראית, וגם את הערך שנמצא בכתובת זו.
  אסור לעשות שימוש כלשהו בכתובת זו, שכן יתכן שערכה משמש את התוכנית שלנו או
  תוכניות אחרות, וכל שינוי שנבצע יביא לנפילת התוכנית שלנו במקרה הטוב, לנפילת
  המחשב במקרה הפחות טוב, ולתוצאות לא נכונות במקרה הגרוע  */
   printf("px = %p, *px = %d\n", px, *px);
```

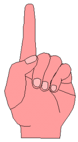
```
/*  נעביר את כתובת המשתנה אל משתנה המצביע */
  px = &x;
  נדפיס את ערכו של המצביע px וניווכח, שערכו הוא למעשה כתובתו  */
  של משתנה x. כלומר, הוא מצביע על משתנה x והערך שנמצא בכתובת
/*  ש-px* מצביע אליה הוא ערכו של משתנה x.
  printf("px = %p, *px = %d\n", px, *px);
/*  נציב ערך בכתובת שאליה מצביע px. כתובת זו היא ערכו של משתנה px */
  *px = 25;
  נדפיס את הערך שנמצא בכתובת ש-px* מצביע אליה  */
/*(כתובתו של משתנה x) ונווכח שערכו של משתנה x שונה, והוא עתה 25.
  printf("x = %d, *px = %d\n", x, *px);
/*  נציב במשתנה y את הערך שנמצא בכתובת ש- px* מתייחס אליה (ערכו של משתנה x) */
  y = *px;
  נדפיס את ערכו של משתנה y וניווכח שערכו שווה לערכו של משתנה x,*/
  כלומר 25. נשים לב, שכתובתם של המשתנים x ו- y לא השתנתה, אך המצביע px
  מוזמן ל"בקר" בכתובתם, לשנות את ערכם, ולעשות "ככל העולה על רוחו".
  מכאן ניתן להבין שמצביע הוא כלי רב עוצמה, אך יחד עם זאת מסוכן ביותר.
/*  לכן יש לנקוט זהירות מירבית בשימוש במצביעים.
  printf("&y = %p , y = %d , *px = %d\n", &y, y, *px);
  printf("Press any key");
  getch();
  return(0);
}
```

תוצאות ריצת התוכנית:

```
x = -29322, y = 12803, *px = -31765
&x = fff4, &y = fff2, &px = fff0
&x = FFF4, &y = FFF2, &px = FFF0
px = 8B95, *px = -31765
px = FFF4, *px = -29322
x = 25, *px = 25
&y = FFF2, y = 25, *px = 25
Press any key
```

בנוסף לאופן המיוחד של ההגדרה, ניתן להשתמש "בערך המוצבע" על ידי המצביע, כפי שמשתמשים בכל משתנה מאותו טיפוס. בהתאם לכך הפעולות הבאות הן חוקיות:

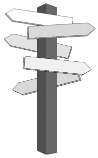
```
(*px)++
f_res = *f_ptr / 8;
```



שים לב!

*משתמשים בסוגריים בפעולת ההגדלה של המצביע: ++(*px). הסיבה לכך נובעת מסדר השערוך של **פקודות יחידניות** (אונריות) כגון: ++, --, *, -, אשר נעשה מימין לשמאל. ללא הסוגריים, הביטוי ++*px היה גורם להגדלת ערכו של המצביע px לאחר גישה למשתנה המוצבע על ידי px. לעומת זאת, אם ברצוננו לבצע **הגדלה מוקדמת** (Pre-increment), אין צורך להשתמש בסוגריים. הפעולה px*++ תגדיל ב-1 את ערך המשתנה המוצבע על ידי px לפני הגישה, ועל-כן לא השתמשנו בסוגריים. מדוע?*

אפשרויות השימוש במצביעים ובאריתמטיקה במשולב הן רבות ומגוונות, אך גם עלולות להיות מקור לטעויות. פעמים רבות, ההבחנה בין המקרים היא דקה, ומומלץ להשתמש בסוגריים כדי להבטיח שאכן יתבצע מה שדרוש.



נקודת ציון!

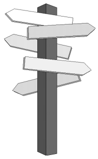
הגדלת ערכו של המצביע בפקודה אונרית, תגדיל אותו בגודל הטיפוס שעליו הוא מצביע. למשל: int יגדל בשני בתים, char יגדל בבית אחד.

בדוגמה הבאה נחלק ב-8 את הערך המוצבע על ידי f_ptr, בהסתמך על הקשר החזק שיש לפעולות & ו-* ביחס לפעולות האריתמטיות *, /, +, -.

לשם כך יש לכתוב:

```
f_res = (*f_ptr) / 8;
```

היתרון הבולט של המצביע הוא הגמישות והדינמיות שבשימוש בו. הגדרת משתנים, כגון: count, sum, num ו-int מייעדת לכל אחד מהם תפקיד ומקום מוגדר בתוכנית, ואין בהם הגמישות הזו.



נקודת ציון!

*מצביע מסוג * void אינו מצביע לדבר, ויכול להכיל מצביע מטיפוס כלשהו. אולם, לא ניתן יהיה לבצע עליו פעולות ++ או --, מכיון שהמהדר אינו "יודע" את גודל הטיפוס המוצבע. שימוש במצביע מסוג זה נעשה ביישומים מתקדמים.*

בעזרת מצביע, ניתן לבצע את רוב הפעולות וההצבות הדרושות בתוכנית.

בדוגמה הבא נציג סימולציה של פקודות אונריות על מצביעים. בסימולציה זו נבחן את התהליכים שלב אחר שלב, ובכל שלב נציג על המסך את ערכיהם של המצביעים ואת כתובותיהם, כך שנוכל להיווכח בשינוי כתובתו או ערכו של מצביע מייד לכשיתרחש.

```
/*****        chp1102.c :קובץ תקליטור        *****/

/*   סימולציה של פקודות אונריות על מצביעים   */

#include <stdio.h>
#include <conio.h>
```

```
/* ----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ------------------------------------------------------*/
int main(void)
 {
   int x, *px;

   clrscr();
      */  עתה נדפיס את ערך המשתנים שהוגדרו וכפי שנראה ערכם אקראי לחלוטין/*
   printf("x = %d , *px = %d\n", x, *px);
      */ נדפיס את כתובת המשתנים שהוגדרו דרך א'. /*
   printf("&x = %x , &px = %x\n", &x, &px);
      */ נדפיס את כתובת המשתנים שהוגדרו דרך ב '. /*
   printf("&x = %p , &px = %p\n", &x, &px);
      /*  נבדוק לאיזו כתובת המצביע px מצביע ומה נמצא בכתובת זו. ניתן לראות
      שהמצביע מצביע לעבר כתובת אקראית, וגם את הערך שנמצא בכתובת זו.
      אסור לעשות שימוש כלשהו בכתובת זו, שכן יתכן שערכה משמש את התוכנית
      שלנו או תוכניות אחרות וכל שינוי, שנבצע יביא לנפילת התוכנית שלנו במקרה
      הטוב, לנפילת המחשב במקרה הפחות טוב, ולתוצאות לא נכונות במקרה הגרוע */
   printf("px = %p, *px = %d\n", px, *px);
      */  נעביר את כתובתו של משתנה למשתנה המצביע  /*
   px = &x;
      /*  נדפיס את ערכו של המצביע px וניווכח, שערכו הוא בעצם כתובתו של
      משתנה x. כלומר, הוא מצביע על משתנה x והערך שנמצא בכתובת עליו px*
      מצביע הוא ערכו של משתנה x */
   printf("px = %p, *px = %d\n", px, *px);
      */ נבצע הצבת ערך בכתובת עליה מצביע px. כתובת זו היא ערכו של משתנה px  /*
   *px = 25;
      /*  נדפיס את הערך שנמצא בכתובת עליו px* מצביע (כתובתו של משתנה x)
      וניווכח, שערכו של משתנה x שונה והוא עתה 25. */
   printf("x = %d , *px = %d\n", x, *px);
      */ נקדם את הערך היושב בכתובת עליו px* מצביע ב-1  /*
   (*px)++;
      /* נדפיס את הערך שנמצא בכתובת ש-px* מצביע אליו (כתובתו של משתנה x)
      וניווכח, שערכו של משתנה x גדל ב- 1 והוא עתה 26. ערכו של
      המצביע לא השתנה, והוא ממשיך להצביע אל המשתנה x */
   printf("x = %d , *px = %d px = %p\n", x, *px, px);
   ++*px;
      /* נדפיס את הערך שנמצא בכתובת ש-px* מצביע אליו (כתובתו של משתנה x)
      וניווכח, שערכו של משתנה x גדל ב- 1 והוא עתה 27. ערכו של
      המצביע לא השתנה, והוא ממשיך להצביע על המשתנה x. */
   printf("x = %d , *px = %d px = %p\n", x, *px, px);
      */ נבצע הגדלה מוקדמת לערך שנמצא בכתובת ש-px* (ב-1 כאמור)  /*
   ++(*px);
```

```
   /*    נדפיס את הערך שנמצא בכתובת ש-px* מצביע אליה (כתובתו של משתנה x)
         וניווכח, שערכו של משתנה x גדל ב- 1 והוא עתה 28. ערכו של
         המצביע לא השתנה, והוא ממשיך להצביע על המשתנה x. הצבת ערכו
         של המצביע בין סוגריים נועדה לצורך בהירות בלבד ועל כן היא מומלצת. */

  printf("x = %d , *px = %d px = %p\n", x, *px, px);
   /*    אזהרה! הגדלה באופן הבא תקדם את הערך שנמצא במצביע, כלומר תקדם
         את הכתובת בשני בתים אל כתובת אשר יתכן והערך שנמצא בה שייך
         למשתנה אחר. כאן מתחיל הנזק. שימו לב! */
  *px++;
   /*    נדפיס את הערך שנמצא במצביע px, כלומר את הכתובת המוצבעת,
         שאינה הכתובת של משתנה x, ואת הערך שנמצא בכתובת זו. */
  printf("*px = %d px = %p\n", *px, px);
  printf("Press any key");
  getch();
  return(0);
}
```

תוצאות ריצת התוכנית:

```
x = -29319, *px = -256
&x = fff4, &px = fff2
&x = FFF4, &px = FFF2
px = 3203, *px = -256
px = FFF4, *px = -29319
x = 25, *px = 25
x = 26, *px = 26 px = FFF4
x = 27, *px = 27 px = FFF4
x = 28, *px = 28 px = FFF4
Press any key
```

מצביעים ומערכים

קיים קשר הדוק בין מצביעים למערכים. המהדר הופך כל התייחסות למערך, כמו למשל array[3]=5, להתייחסות באמצעות מצביע ו**היסט** (Offset), ומקבלים שהמצביע הוא שם המערך. ההבדל בין משתנה שהוא מצביע לבין שם של מערך, הוא בכך שהאחרון קבוע, ולכן בלתי ניתן לשינוי כפי שמאפשר מצביע. את משמעות ההיסט נסביר בהמשך.

לפנינו דוגמה שתמחיש את ההקבלה בין שם מערך לבין מצביע:

```
(1)  char c_array[5], *c_ptr;
(2)  c_ptr = c_array;
(3)  c_array[4] = 'a';
(4)  *(c_ptr+4) = 'a';
```

הסבר:

א. במשפט 2 הורינו כי c_ptr יצביע לתחילת המערך. במשפט c_ptr = c_array מומחשת ההקבלה שקיימת בשפת C בין שמות מערכים לבין מצביעים. יש להדגיש שהצבה הפוכה c_array = c_ptr אינה אפשרית שכן, כפי שציינו קודם, שם המערך **הוא קבוע**.

ב. משפטים 3 ו-4 מבצעים את אותה הפעולה בדיוק: הצבת האות a לאיבר **האחרון** של המערך c_array. כזכור, בשפת C מסמנים את האיבר הראשון ב-0, ולכן האיבר החמישי יסומן כך: c_array[4].

את קטע התוכנית שלמעלה ניתן לשכתב באופן הבא:

```
c_ptr = &c_array[0];
*(c_array+4) = 'a';
*(c_ptr+4) = 'a';
```

כפי שרואים, כתובת האיבר הראשון במערך וערכו של שם המערך **זהים**. עובדה זו נובעת מההגדרה "**שם המערך הוא מצביע לאיבר הראשון בו**". כתוצאה מהקבלה זו, ההצבה ;'a' = (c_array+4)* היא חוקית. המהדר מתייחס תמיד לביטוי x[i] כמו אל ‎*(x+i).

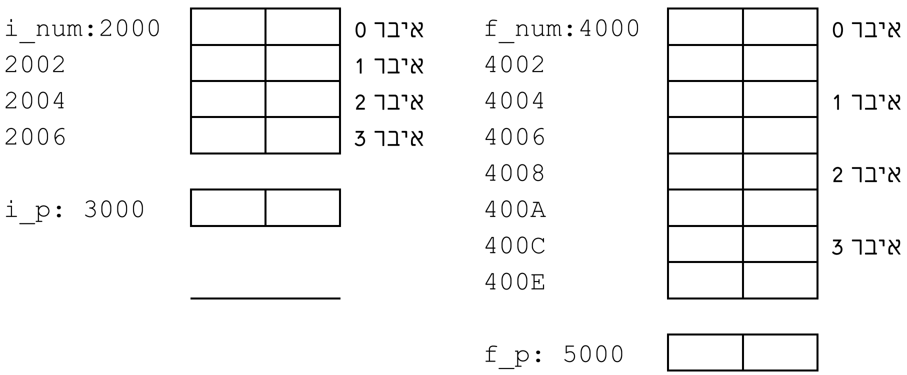
כאן המקום להסביר את הקשר בין טיפוס המערך/מצביע לבין חישובי ההיסט. נניח שנתונים שני מערכים, אחד מטיפוס שלם (int) והשני מטיפוס ממשי (float).

נגדיר:

```
int i_num[4], *i_p;
float f_num[4], *f_p;
```

המשתנים i_p ו-f_p הם מצביעים מטיפוס שלם וממשי בהתאמה.

תמונת זיכרון **אפשרית** שמתקבלת לאחר הכרזה זו מוצגת באופן סכימטי בתרשים 11.3:



תרשים 11.3: תמונת זיכרון מקורית של שני מערכים

גודלו של המצביע בבתים הינו קבוע ותלוי במרחב הזיכרון במחשב. נניח, שגודל הטיפוס השלם הוא 2 בתים, וגודל הטיפוס הממשי הוא 4 בתים. בנוסף נניח שבמחשב שלנו מרחב הזיכרון הוא 2^{16} ולכן גודל המצביע הוא שני בתים, כדי להכיל את

הכתובת העליונה של מרחב הזיכרון. כעת נבצע סדרת פעולות זהה על 2 המערכים ונראה את ההבדלים:

א.
```
i_p = i_num;              /*  i_p = 2000 */
*(i_p+2) = 5;             /* איבר שלישי מקבל 5 */
```

ב.
```
f_p = f_num;              /*  f_p = 4000 */
*(f_p+2) = 5;
```

במקרה א', המהדר רואה את i_p כמצביע מטיפוס שלם. לכן, לפני חיבור ההיסט לערכו של המצביע הוא יכפול אותו בשניים, שכן זהו גודלו של הטיפוס השלם. כתובת התוצאה היא: 2 * 2 + 2000, כלומר 2004. לכתובת זו מציבים את הערך 5, וכפי שניתן לראות בתרשים 11.3, האיבר השלישי (איבר 2) אכן נמצא בכתובת 2004.

במקרה ב', המהדר רואה את f_p כמצביע מטיפוס ממשי שגודלו 4 בתים ולכן, לפני חיבור ההיסט, הוא יכפילו ב-4. כתובת התוצאה היא 4008, וזהו המיקום הנכון של האיבר השלישי (איבר 2) במערך f_num.

לאחר ההצבה נקבל את תמונת הזיכרון, אשר מוצגת בתרשים 11.4.

2000		
2002		
2004	05	00
2006		
3000	00	20

4000		
4002		
4004		
4006		
4008	00	00
400A	00	05
400C		

5000	00	40

תרשים 11.4: *תמונת זיכרון של מערכים במהלך העבודה*

הערך 5 בייצוג ממשי אינו 000000005 (בתאים 4008 עד 400B) כפי שמובא כאן להקלת ההסבר, אלא ייצוג מעריכי, שפרטיו נקבעים בכל מחשב בצורה שונה.

אם היינו משתמשים ב-i_p גם עבור ההצבה במערך הממשי f_num, המהדר היה מחשב את ההיסט לפי הטיפוס השלם, והייתה הצבה לכתובת 4004 (2 * 2 + 4000). זוהי כתובת האיבר השני במערך f_num, ולא האיבר השלישי כפי שרצינו. דוגמה זו ממחישה מדוע יש להקפיד על צורת השימוש הנכונה במצביעים.

אריתמטיקה של מצביעים

בדוגמה קודמת הצגנו אפשרות להוסיף או לחסר קבועים ממצביעים. התרגום של פעולות אלו לפעולות הכפל המתאימות נעשה על ידי המהדר. מכאן נגזרת היכולת לקדם, או להקטין מצביעים בעזרת פעולות ++, --, כפי שנראה בהמשך.

הפעולות האפשריות על מצביעים הן מועטות: אין משמעות לכפל או חילוק של מצביעים. פעולות לוגיות כגון <, ==, > מוגבלות למצביעים המתייחסים לאותו מערך וכל חריגה תגרום לתוצאות לא צפויות. כאשר נכיר טוב יותר את השפה, נלמד לעקוף מגבלות אלו ולהשתמש באריתמטיקה של מצביעים גם בפעולות אריתמטיות-לוגיות

כגון ^,¦, &. שימוש כזה מחייב הכרה טובה של מפת הזיכרון ושימוש זהיר בהצבות שינבעו מפעולות אלו.

נראה מספר דוגמאות פשוטות שתמחשנה את האפשרויות שכבר הכרנו:

```
            /*****      chp1103.c  :קובץ תקליטור       *****/

      #include <stdio.h>

(1)   int numbers[20], *n_ptr, i ;
(2)   char *ptr, *c, name[6]="DAVID" ;

(3)   void main(void)
(4)   {
(5)   for (i=0, n_ptr=numbers ; i<20 ; ++i, ++n_ptr)
(6)        *n_ptr = 0 ; /* numbers[] אתחול מערך */

(7)   ptr = c = name;
(8)   while (*(ptr++))
         ;        /* הצבע את ptr לסוף המחרוזת */
(9)   if (ptr > c)
(10)  printf("ptr is greater than c - the difference is
               %d\n" ,ptr - c) ;
(11)  }
```

לאחר הרצת התוכנית יתקבל הפלט הבא:

```
ptr is greater than c -the difference is 6
```

הסבר:

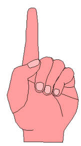
א. בשורות 2-1 אנו מגדירים את המשתנים והמצביעים ומאתחלים את המערך name במחרוזת "DAVID".

ב. שורות 6-5 הן דוגמה לאתחול מהיר של מערך בעזרת לולאת for ומצביע מטיפוס שלם, n_ptr. צורת אתחול זו שימושית כאשר צריך לאתחל מערכים דינמיים.

ג. השימוש במונה i בשורות 6-5 מיותר, וניתן היה לנקוט בדרך שונה:

```
for(n_ptr=numbers;(n_ptr-numbers)<20;*(n_ptr++)=0);
```

כפי שהסברנו קודם, ניתן לבצע פעולות חיסור/חיבור כאשר המצביעים מצביעים לאותו מערך. כאן נעזרנו בעובדה זו ואת פעולת ההצבה הכנסנו במסגרת שלב העדכון של משתנה הלולאה במשפט for.



שים לב!

פעולת החיסור (n_ptr_number) היא ההפרש בין שתי כתובות בזיכרון. התוצאה היא מספר, אשר במקרה שלנו מוכנס לתבנית מספרית (%d) מסוג integer.

ד. במשפט while בשורה 8 הסתמכנו על העובדה, שהאיבר האחרון במחרוזת הוא 0 (null) בסוף קבוע מחרוזת) ומשמש כתנאי עצירה. בסיום הלולאה, ptr יצביע לקצה המערך []name. פעולת "קידום אחרי" הוכנסה לסוגריים, כדי למנוע פעולה לא רצויה של הגדלת הערך המוצבע על ידי המצביע, במקום ערך המצביע עצמו.

ה. הערך שמודפס במשפט if (בשורות 9-10) הוא האורך של המערך name, מכיון ש-ptr מצביע לסוף המערך ו-c מצביע לתחילתו.

מצביעים וערכי פונקציות

העברת פרמטרים לפונקציות נעשית בשפת C "**לפי-ערך**" (By-value). נניח שברצוננו לכתוב פונקציה, שתקבל שני ערכים מטיפוס שלם ותחזיר את המכפלה ואת המנה שלהם.

נכתוב תוכנית להצגת אפשרות זו:

```
      /*****      chp1104.c  :קובץ תקליטור        *****/

      #include <stdio.h>

(1)   void multdiv (int x, int y)
(2)   /* גוף הפונקציה */
(3)   {
(4)        int mult, div ;

(5)   mult = x * y ;
(6)   div = x / y ;
(7)   x = mult ;
(8)   y = div ;
(9)   }

(10)  void main(void)
(11)  {
(12)       int  x, y ;

(13)  x = 16;
(14)  y = 4;
(15)  multdiv(x, y);
(16)  printf ("x*y is %d, x/y is %d\n",x,y);
(17)  }
```

הסבר:

א. לא ניתן להחזיר בפונקציה אחת יותר מערך אחד במשפט return. לכן ניסינו לפתור את הבעיה על ידי הצבת תוצאות הכפל והחילוק למשתנים x ו-y (שורות 7, 8).

ב. לאחר הקריאה לפונקציה ()multdiv הדפסנו את ערכי המשתנים x ו-y שאמורים להכיל את התוצאות, אך להפתעתנו (הלא רבה) קיבלנו את המשפט הבא:

```
x*y is 16, x/y is 4
```

ג. הסיבה לתוצאה זו היא תכונת העברת הערכים. הפונקציה ()multdiv אינה "מכירה" את המשתנים x,y המוכרזים בתוכנית הראשית ()main. כאשר אנו קוראים לה, מועתקים ערכיהם (4, 16) למשתנים **מקומיים זמניים** המוכרזים בפונקציה ()multdiv. כל הפעולות המתבצעות על אותם משתנים זמניים, אינן משפיעות על המשתנים שב-()main.

ד. פתרון חלקי למצב אפשרי על ידי הגדרת x,y כמשתנים גלובליים מחוץ ל-()main, וקריאה ל-()multdiv ללא ערכים כלל. במקרה כזה היו התוצאות המודפסות נכונות, אך שכרנו יצא בהפסדנו: הפונקציה ()multdiv יכולה לבצע את הפעולות **רק בין שני משתנים מסוימים - x,y**.

ה. השימוש במצביעים מאפשר לעקוף בעיה זו ולשנות מספר בלתי מוגבל של משתנים מתוך פונקציה כלשהי.

המצביע, כשמו, מצביע אל מקום בזיכרון. **לאותו מקום** יכולים להצביע מצביעים אחדים וכאן מצוי הפתרון:

```
      /*****      chp1105.c  :קובץ תקליטור      *****/

      #include <stdio.h>

(1)   void multdiv(int *px, int *py)
(2)   {
(3)     int  mult, div ;

(4)     mult = *px * *py ;
(5)     div = *px / *py ;
(6)     *px = mult;
(7)     *py = div;
(8)   }

(9)   main()
(10)  {
(11)    int  x, y ;

(12)    x = 16;
(13)    y = 4;
(14)    multdiv(&x, &y);
(15)    printf ("x*y is %d, x/y is %d\n", x,y);
(16)  }
```

הסבר:

א. שינינו את הפונקציה ()multdiv. הערכים שהיא מקבלת הם עתה מצביעים מטיפוס שלם.

ב. הקריאה לפונקציה ()multdiv נעשית עם כתובותיהם של x ו-y כפרמטרים. הסיבה לכך היא, שהפרמטרים הפורמליים של הפונקציה הם מצביעים. בזמן הקריאה מקבלים px ו-py את ערכי הכתובות בהתאם והם מצביעים לתוכנם של המשתנים x ו-y בהתאמה.

ג. פעולות ההצבה בשורות 6, 7 משנות את הערכים של x ו-y. זוהי גישה עקיפה אך יעילה לפתרון הבעיות שהעלינו קודם.

תוכנית הסימולציה הבאה תמחיש בפנינו את פעולת הפונקציה ;()multdiv.

```
         /*****      chp1106.c :קובץ תקליטור       *****/

/* multdiv() סימולציה של פונקצית  */

#include <stdio.h>
#include <conio.h>

/* (i) integer (ptr) pointer מצביע למשתנה שלם ועל כן שמו  */
void multdiv(int *i_ptr_x, int *i_ptr_y);

/* ----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ------------------------------------------------------*/
int main(void)
 {
  int x = 16, y = 4;

  clrscr();
  printf("(הגדרה)(x כתובת) &x = %p, (x ערך) x = %d ,"
      "(y כתובת) &y = %p, (y ערך) y = %d\n", &x, x, &y, y);
  multdiv(&x, &y);
  printf("\n\nx * y is %d, x/y is %d\n", x, y);
  printf("\nPress any key\n");
  getch();
  return(0);
 }
/* ----------------------------------------------------------
 ;  F U N C T I O N S
 ; ------------------------------------------------------*/
void multdiv(int *i_ptr_x, int *i_ptr_y)
 {
  int mult, div;
```

```
  printf("\n(הגדרה)(mult כתובת) &mult = %p, (mult ערך)"
         "mult = %d ,\n(div כתובת) &div = %p, (div ערך)"
         "div = %d\n",&mult, mult, &div, div);
  printf("\n(כתובת המצביע) &i_ptr_x = %p,"
         "(הכתובת המוצבעת) i_ptr_x = %p ,\n"
         "(ערך המצביע) *i_ptr_x = %d,"
         "\n(כתובת המצביע) &i_ptr_y = %p,"
         "(הכתובת המוצבעת) i_ptr_y = %p ,\n"
         "(ערך המצביע) *i_ptr_y = %d,",
         &i_ptr_x, i_ptr_x, *i_ptr_x,
         &i_ptr_y, i_ptr_y, *i_ptr_y);
   /*    יש לשים לב שכתובת המצביע שונה מכתובת המשתנה מכיון ששניהם משתנים
         התופסים מקום בזיכרון. הערך היושב במצביע הוא הכתובת של המשתנה ולכן
         כל פעולה אשר תתבצע על תוכן כתובת זו כאילו התבצעה על המשתנה עצמו */
  mult = (*i_ptr_x) * (*i_ptr_y);
  div  = (*i_ptr_x) / (*i_ptr_y);
  *i_ptr_x = mult;
  *i_ptr_y = div;
 }
```

תוצאות ריצת התוכנית:

```
(הגדרה)(x כתובת) &x = FFF4, (x ערך) x = 16 ,(y כתובת) &y = FFF2, (y ערך) y = 4

(הגדרה)(mult כתובת) &mult = FFE8, (mult ערך)mult = 168 ,
(div כתובת) &div = FFE6, (div ערך)div = 741

(כתובת המצביע) &i_ptr_x = FFEE,(הכתובת המוצבעת) i_ptr_x = FFF4 ,
(ערך המצביע) *i_ptr_x = 16,
(כתובת המצביע) &i_ptr_y = FFF0,(הכתובת המוצבעת) i_ptr_y = FFF2 ,
(ערך המצביע) *i_ptr_y = 4,

x * y is 64, x/y is 4

Press any key
```

מצביעים למחרוזות ושימושים בפונקציות

בעת הדיון במערכים הזכרנו, כי השימוש במצביעים יפשט את התוכניות ויעשה אותן ברורות יותר. נזכיר, שאנו מניחים כי מחרוזות מסתיימות ב-null (0), ועובדה זו משמשת בסיס לכל הפעולות עליהן בעזרת מצביעים. כעת נוכל לשכתב את ארבע הפונקציות העיקריות העוסקות במחרוזות.

פונקציה למדידת אורך של מחרוזת

```
    /*****     chp1107.c :מתוך קובץ תקליטור     *****/
(1)  int strlen(char *s)
(2)  {
(3)    char *s_ptr;
(4)    for (s_ptr=s ; *s_ptr ; ++s_ptr)
         ;
(5)    return(s_ptr - s);
(6)  }
```

הסבר:

א. הערך שמקבלת הפונקציה ()strlen הוא מצביע לטיפוס תו, המצביע לתחילת המחרוזת שאת אורכה מודדים.

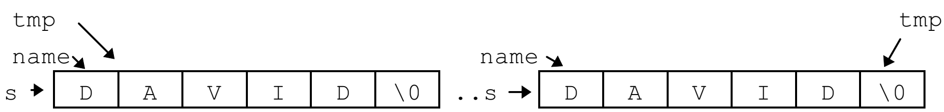
ב. בלולאת for, בשורה 4, מציבים למצביע s_ptr את ערכו של s ולכן s_ptr מצביע לראשיתו של המערך. כאשר s_ptr מגיע לסוף המערך, יהיה הערך המוצבע על ידו 0 (אפס) ולולאת for תסתיים.

ג. בשורה 5 מוחזר הפרש הכתובות שבין המצביע s_ptr לבין s, אשר מצביעים לתחילת ולסוף המחרוזת. ערך זה הוא אורכה של המחרוזת.

דוגמה:

```
char *name = "DAVID";
int  i;
i = strlen(name);  /* i=5 */
```

מהלך שישי - לולאת for . . . **מהלך ראשון - לולאת for**



תרשים 11.5

למעשה, ניתן לוותר בפונקציה זו על המשתנה s_ptr. מכיון שהמשתנה s הינו משתנה מקומי וזמני, ניתן לשנות את ערכו בתוך הפונקציה, מבלי שהדבר ישפיע על המשתנה בו השתמשו כאשר קראו לפונקציה:

```
strlen(s)
  char *s ;
{
  int i;
  for (i=0; *s++; ++i) ;
  return(i) ;
}
```

שים לב!

הרווח בכך שולי, כי היה עלינו להציג את המשתנה i כמונה.

תוכנית הסימולציה הבאה תמחיש את פעולת הפונקציה למדידת אורך של מחרוזת.

```
        /*****      chp1108.c :קובץ תקליטור      *****/

/* סימולצה של פונקציה למדידת אורך מחרוזת */

#include <stdio.h>
#include <conio.h>

int a_strlen(char *s);

/* -----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ---------------------------------------------------------*/
int main(void)
 {
   char *s = "Hod-Ami";

   clrscr();
   /* נדפיס את כתובתה וערכה של המחרוזת שהוגדרה */
   printf("(הגדרה)(כתובת המחרוזת)  &s = %p,"
          "(הכתובת המוצבעת)  s(&->)  = %p ,\n"
          "(הערך היושב בכתובת המוצבעת)  s = %s\n\n", &s, s, s);
   printf("strlen(%s) = %d\n", s, a_strlen(s));
   printf("Press any key\n");
   getch();
   return(0);
 }
/* -----------------------------------------------------------
 ;  F U N C T I O N S
 ; ---------------------------------------------------------*/
int a_strlen(char *s)
 {
   /* (s) string (ptr) pointer מצביע למחרוזת ועל כן שמו */
   char *s_ptr;

   for (s_ptr = s; *s_ptr; s_ptr++)
   printf("(for)(כתובת המצביע)  &s_ptr = %p,"
          "(הכתובת המוצבעת)  s_ptr(&->)  = %p ,\n"
          "(הערך היושב בכתובת המוצבעת)  *s_ptr = %c\n", &s_ptr,
                                                  s_ptr, *s_ptr);
```

```
  /*   ?מה הכתובת של המשתנים כעת  */
  printf("\n(after for)(כתובת המצביע) &s_ptr = %p,"
         "(הכתובת המוצבעת) s_ptr(&->) = %p ,\n"
         "(כתובת המחרוזת) &s = %p,"
         "(הכתובת המוצבעת) s(&->) = %p\n", &s_ptr,
                                            s_ptr, &s, s);
  /* ? מהו הפרש הכתובות  */
  printf("\n(הכתובת המוצבעת) s_ptr(&->) = %p (-) "
         "(הכתובת המוצבעת) s(&->) = %p (=) %d"
         "\n", s_ptr, s, s_ptr - s);
  return(s_ptr - s);
 }
```

תוצאות ריצת התוכנית:

הערה: *הכתובת של המחרוזת s יכולה להשתנות על פי ההקצאה של מערכת ההפעלה. יש לשים לב לשינוי הכתובת ולא לערך הספציפי.*

```
(הגדרה)(כתובת המחרוזת) &s = FFF4,(הכתובת המוצבעת) s(&->) = 00AA ,
(הערך היושב בכתובת המוצבעת) s = Hod-Ami

(for)(כתובת המצביע) &s_ptr = FFEC,(הכתובת המוצבעת) s_ptr(&->) = 00AA ,
(הערך היושב בכתובת המוצבעת) *s_ptr = H
(for)(כתובת המצביע) &s_ptr = FFEC,(הכתובת המוצבעת) s_ptr(&->) = 00AB ,
(הערך היושב בכתובת המוצבעת) *s_ptr = o
(for)(כתובת המצביע) &s_ptr = FFEC,(הכתובת המוצבעת) s_ptr(&->) = 00AC ,
(הערך היושב בכתובת המוצבעת) *s_ptr = d
(for)(כתובת המצביע) &s_ptr = FFEC,(הכתובת המוצבעת) s_ptr(&->) = 00AD ,
(הערך היושב בכתובת המוצבעת) *s_ptr = -
(for)(כתובת המצביע) &s_ptr = FFEC,(הכתובת המוצבעת) s_ptr(&->) = 00AE ,
(הערך היושב בכתובת המוצבעת) *s_ptr = A
(for)(כתובת המצביע) &s_ptr = FFEC,(הכתובת המוצבעת) s_ptr(&->) = 00AF ,
(הערך היושב בכתובת המוצבעת) *s_ptr = m
(for)(כתובת המצביע) &s_ptr = FFEC,(הכתובת המוצבעת) s_ptr(&->) = 00B0 ,
(הערך היושב בכתובת המוצבעת) *s_ptr = i

(after for)(כתובת המצביע) &s_ptr = FFEC,(הכתובת המוצבעת) s_ptr(&->) = 00B1 ,
(כתובת המחרוזת) &s = FFF2,(הכתובת המוצבעת) s(&->) = 00AA

(הכתובת המוצבעת) s_ptr(&->) = 00B1 (-) (הכתובת המוצבעת) s(&->) = 00AA (=) 7
strlen(Hod-Ami) = 7
Press any key
```

פונקציה להעתקת מחרוזת

```
        /*****     chp1107.c :מתוך קובץ תקליטור      *****/
(1)    strcpy(char *target, char *source)
(2)    {
(3)      while (*(target++) = *(source++))
              ;
(4)    }
```

הסבר:

א. הפונקציה ()strcpy להעתקת מחרוזת מקבלת כערכים שני מצביעים מטיפוס תו. מניחים שהמצביע source מצביע למחרוזת כלשהי המסתיימת ב-null, ואילו המצביע target מצביע למקום פנוי בזיכרון, שגודלו לפחות כאורכה של המחרוזת ובתוספת מקום לצורך null.

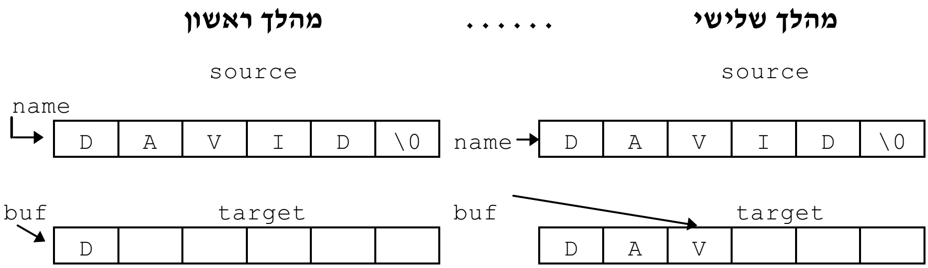
ב. הפונקציה ()strcpy מורכבת ממשפט while בודד הכולל הן את העתקת המחרוזת והן את הבדיקה לסיום. זוהי המחשה לעוצמתה של השפה בעת שימוש מושכל במצביעים. הערך של משפט השמה הוא ערכו של הביטוי מימין לסימן ההשמה ועל עובדה זו אנו מסתמכים כתנאי שיעצור את לולאת while. בהגיענו ל-null המסיים, יהיה ערכו של הביטוי 0 (זהו ערכו של האיבר האחרון), הביטוי יהיה שקר והלולאה תסתיים.

ג. שוב עלינו להתריע מפני מצב, שבו המצביע target מצביע למקום קטן מדי, או שאינו מצביע למקום פנוי כלל. במצב כזה התוצאות אינן צפויות ומהלך התוכנית עלול להשתבש.

ראה תיאור של פעולת העתקה בתרשים 11.6.

דוגמה:

```
char buf[6];
strcpy(buf, name);
```



***תרשים 11.6:** העתקת מחרוזת*

פונקציה להשוואת מחרוזות

```
     /*****       chp1107.c  :מתוך קובץ תקליטור       *****/

(1)    strcmp (char *first, char *second)
(2)    {
(3)    for (; *first==*second && *first && *second;
                                 ++first, ++second);
(4)    return (*first - *second);
(5)    }
```

הסבר:

א. לולאת for נמשכת כל עוד המצביעים first ו-second מצביעים לערכים זהים, ואינם שווים לאפס.

ב. בגמר כל השוואה, המצביעים מקודמים לקראת ההשוואה הבאה.

ג. בנקודת אי-השוויון הראשונה, או כאשר אחת מהמחרוזות הגיעה לסיומה, נפסקת לולאת for. התוצאה המוחזרת היא ההפרש בין ערכי ASCII של התווים בנקודת ההפסקה.

פונקציה לצירוף מחרוזות

```
    /*****      chp1107.c :מתוך קובץ תקליטור          *****/

(1)   void strcat (char *base, char *add)
(2)   {
(3)   char *s_ptr, *c;

(4)   for (s_ptr=base; *s_ptr; ++s_ptr);
           ;
(5)   for (c=add;s_ptr = *c; ++c, ++s_ptr);

(6)   }
```

הסבר:

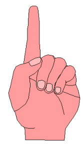
א. יש לצרף את המחרוזת המוצבעת על ידי המצביע add, להמשכה של המחרוזת המוצבעת על ידי המצביע base.

ב. בשורה 4 מקדמים את המצביע s_ptr במסגרת לולאת for, עד סוף המחרוזת base.

ג. בשורה 5 מועתקים הערכים של המחרוזת הנוספת המוצבעים על ידי המצביע c, אל המקומות המוצבעים על ידי המצביע s_ptr.

בפונקציה ()strcpy הצגנו אפשרות תכנות ללא המשתנה s_ptr. בפונקציה ()strcat, נוכל להציג אפשרות חסכון דומה במשתנים s_ptr ו-c. במקרה זה, יהיה החסכון משמעותי יותר, שכן אין צורך להציג משתנים נוספים:

```
strcat (char *base, char *add)
{
     while (*base)
        ++base;
     while (*(base++) = *(add++))
          ;
}
```



שים לב!

בכל הפונקציות שראינו לטיפול במחרוזות השתמשנו בתבנית פונקציה כזו:

```
funct_name(char *var_1, char *var_2)
```

תבנית קודמת היתה:

```
funct_name (var_1, var_2)
  char *var_1, var_2
...
```

כלומר, השוני הוא בכך שהגדרת המשתנים נעשית בהגדרת הפונקציה ולא בשורה נפרדת. תוכל לבחור בדרך שרצויה לך.

מערכים של מצביעים

עד כה התייחסנו למצביעים כמשתנים בודדים, המצביעים לכתובת כלשהי בזיכרון. כשם שיש מערכים של משתנים מהטיפוסים שהכרנו עד כה, ניתן להגדיר גם מערך של מצביעים. בכך אפשר להתייחס לכמות גדולה של נתונים במרוכז ובגמישות, כפי שנראה בדוגמה הבאה.

נניח שברצוננו לרכז שורה של הודעות, אך איננו יודעים לכמה מהן נזדקק, ומה יהיה האורך של כל אחת מהן. אם נשתמש במנגנון המערכים שהכרנו עד כה, יהיה עלינו לקבוע מראש את מספר ההודעות המירבי ואת האורך המירבי הצפוי להודעה אחת, ולציין זאת מראש בהגדרת המערך.

בדוגמה שלפנינו נגדיר מערך שבו עד 100 הודעות שהאורך המירבי של כל אחת מהן הוא 50 בתים:

```
char messages[100][50]=
                       {
"hello everybody",
"this is the second line",
"this line is the longest and that's why length=50",
"short line",
"last message",
                      };
```

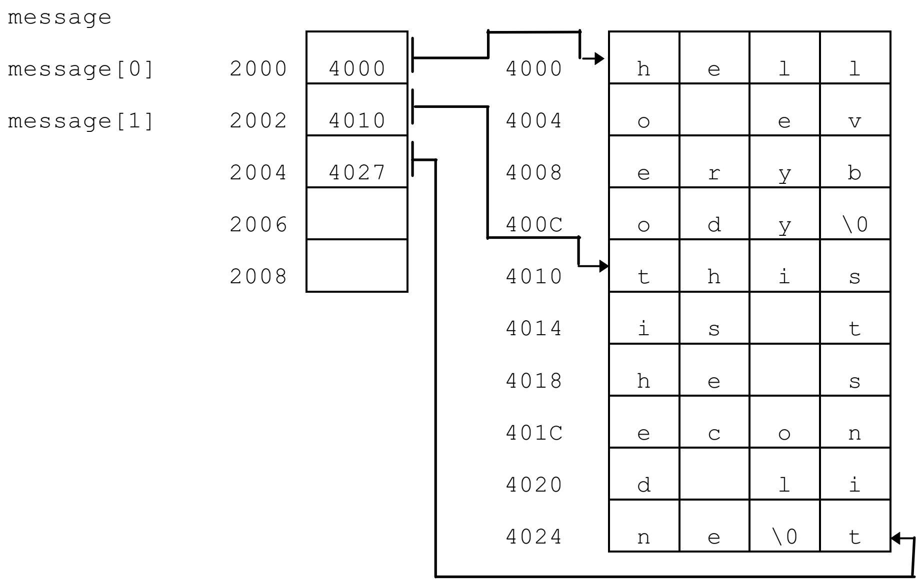
הידור הקטע הזה יגרום להקצאת שטח זיכרון שגודלו 100*50, שהם 5000 בתים. אם נרצה להוסיף הודעה שאורכה גדול מ-49 בתים (כזכור, יש לשמור תו אחד לתו הסיום (null), או להגדיר יותר מ-100 הודעות, יהא עלינו לשנות את הגדרת המערך בהתאם ולהדר שוב את התוכנית. ניתן לראות שניצול המקום בזיכרון בדוגמה זו אינו יעיל, כי במקרה זה אנו זקוקים למקום עבור 5 הודעות בלבד ורובן קצרות ואינן מנצלות את 50 המקומות שהוקצו לכל אחת מהן.

ההגדרה הבאה תאפשר לנו למנוע את החסרונות האלה, ולנצל טוב יותר את הזיכרון העומד לרשותנו:

```
char  *message[]={
                  "hello everybody",
                  "this is the second line",
                  "this line is the longest...",
                  "short....",
                  "last message"
                  } ;
```

המערך message הוגדר כ**מערך של מצביעים**. כל אחד מאיבריו הוא מצביע מטיפוס **תו** (char), אשר מצביע על מחרוזת הנמצאת בזיכרון. ניתן לראות שבמקרה זה אין צורך בכלל להגדיר את אורך ההודעה הארוכה ביותר. בזמן ההידור של התוכנית, המהדר מקצה לכל הודעה את כמות הזיכרון הדרושה לה, וכל מצביע יקבל את הערך שלו בהתאם לכך.

תמונה אפשרית של הזיכרון לאחר ביצוע ההידור ניתנת בתרשים 11.7.



***תרשים 11.7:** מערך של מצביעים*

הדפסת הודעה המבוססת על המשפטים האלה תתבצע באופן דומה בשתי צורות ההגדרה. לדוגמה, ביצוע המשפט השני:

```
printf("%s", message[1]);
```

ובדרך השנייה:

```
printf("%s\n", *(message+1));
```

שים לב!

message - שם המערך הוא גם כתובתו וכאשר נוסף לו 1 נגיע לכתובת הבאה במערך.

** - מציין את הערך שנמצא בכתובת זו.*

התוצאה:

```
this is the second line
```

השימוש במערכים של מצביעים מאפשר גמישות ויעילות רבים יותר כאשר עוסקים במספר לא ידוע של משתנים, שגם האורך של כל אחד מהם אינו ידוע מראש.

מצביעים למצביעים

מדוע להסתפק במצביעים פשוטים בלבד? האם יש אפשרות להגדיר מצביע שיצביע למערך של מצביעים?

שפת C מאפשרת הגדרה כזו. לדוגמה,

```
int **ptr, *numbers[10];
ptr = numbers;
```

נוכל להתייחס למספר השלם המוצבע על ידי המצביע המצוי ב-numbers[0] בדרך זו:

```
i = *numbers[0];
```

או:

```
i = **ptr;
```

למספר השלם המוצבע על ידי המצביע שב-numbers[3] נתייחס כך:

```
i = *numbers[3];
```

או:

```
i = **(ptr+3);
```

למעשה, אין גבול ל**רמת העקיפין** (Indirection level) בה ניתן להשתמש בתוכניות. כל רמה כזו מוסיפה מרחב פעולה ועוצמה גדולים יותר מאשר הרמה הקודמת. הבעיה בשימוש ברמות עמוקות יותר היא המעקב אחרי מהלך התוכנית והסיכון הרב לשגיאות. רמת עקיפין שנייה (מצביע למצביע) מאפשרת פתרון יעיל וקצר של בעיות מסוימות ואפשר להסתפק בה ברוב המקרים.

נניח שברצוננו לחשב את ממוצעי הציונים של התלמידים בבתי הספר. כל בי״ס מיוצג על ידי מערך של מצביעים מטיפוס שלם, המייצגים את כיתות ביה״ס. כל כיתה מיוצגת על ידי מערך של שלמים (בעל גודל שאינו קבוע מראש). תחילתו של כל מערך היא ציוניהם של תלמידי הכיתה והאיבר האחרון יכיל את המספר 1-. הפונקציה שתחשב את ממוצעי הציונים תקבל ערך אחד בלבד - את שם ביה״ס, ותחזיר את ממוצע הציונים של כל התלמידים באותו ביה״ס. במהלך החישוב יודפס הציון הממוצע של כל כיתה.

התוכנית:

```
      /*****      chp1109.c  קובץ תקליטור:      *****/

     #include <stdio.h>

(1)  int      classA1[5]={95,76,27,42,-1},
              classA2[7]={100,80,77,12,0,90,-1},
(3)           classA3[6]={20,55,90,82,100,-1},
(4)           classA4[6]={76,64,88,99,85,-1} ;

(5)  int      *school_A[] =
                     {classA1,classA2,classA3,classA4,0};

(6)  int      classB1[7]={72,77,88,95,100,50,-1},
(7)           classB2[7]={58,79,89,91,95,25,-1},
(8)           classB3[8]={77,79,89,91,95,92,25,-1};

(9)  int      *school_B[] = {classB1,classB2,classB3,0};

(10) int      tot_num, tot_sum;

(11) float average_grade(int** school)
(12)
(13) {
(14)   int   *student_grade, num, sum, class_num = 0;

(15)   tot_num = tot_sum = class_num = 0 ;
(16) while (student_grade = *school)
          {
(17)      num=sum=0;
(18)      while (*student_grade != -1)
              {
(19)          ++num;
(20)          sum+= *student_grade;
(21)          ++student_grade;
(22)          }
(23)      printf("Average of class no. %d is %f\n",
                          ++class_num,(float) (sum/num));
(24)      tot_sum += sum;
(25)      tot_num += num;
(26)      ++school;
(27)      }
(28)   return(tot_sum/tot_num);
(29) }
```

```
(30) main()
(31) {
(32)   float avrg;

(33)   avrg = average_grade (school_A) ;
(34)   printf("SCHOOL A had %d student with an Average of
                                   %f\n", tot_num,avrg);
(35)   avrg = average_grade (school_B) ;
(36)   printf("SCHOOL B had %d student with an Average of
                                   %f\n", tot_num,avrg);
(37) }
```

הסבר:

א. השורה 11 מגדירה את סוג הפונקציה וערכיה. במקרה זה, מוגדרת הפונקציה average_grade() (ציון-ממוצע) מטיפוס ממשי, כבעלת ערך יחיד school, שהוא מצביע למצביע מטיפוס שלם.

ב. נבחן עתה בתרשים 11.8 תמונת זיכרון אפשרית על-פי ההגדרות של בתי הספר. הגדרת הערך school כמצביע למצביע מאפשרת גישה לבי״ס שלם באמצעותו, כפי שמודגם בקווים המרוסקים.

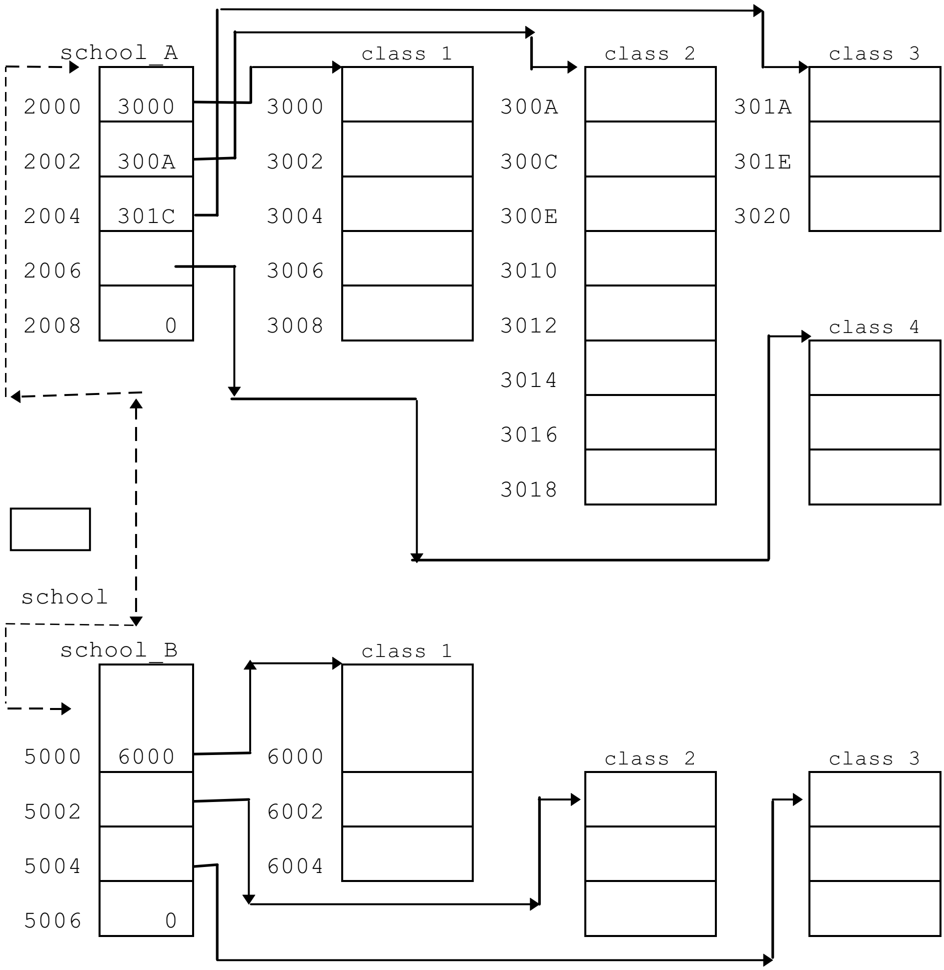
ג. בשורה 14 מוגדרים המשתנים המקומיים, בהם תשתמש הפונקציה במהלך חישוביה. המצביע student_grade יצביע בכל פעם על הציון הבא, אותו נוסיף לסה״כ. tot_sum, tot_num יכילו את מספר התלמידים ואת סיכום ציוניהם בכל הכיתות. sum ו-num הם מוני ביניים של ערכים אלה.

ד. שורות 16-28 מכילות את הלולאה העיקרית המבוצעת בכל שלב של מעבר על ציוני התלמידים בכיתה כלשהי. היא מדפיסה את תוצאת הביניים כנדרש.

ה. ההצבה בפקודה while בשורה 16 מעידה על היחס של רמות העקיפין שקיימות בין המצביעים student_grade ו-school.

ו. בשורה 28 מוחזר הציון הממוצע המשוקלל לפי מספר התלמידים וסיכום ציוניהם.

ז. שים לב לתנאי העצירה בשתי הלולאות. תנאי העצירה של הלולאה החיצונית מיוצג למעשה על ידי משפט ההצבה (student_grade = *school) אשר, כפי שהזכרנו בפרקים קודמים, ערכו הוא ערך ההצבה (*school). תנאי העצירה של הלולאה הפנימית (שורה 18) הוא הביטוי הבוליאני (*student_grade != -1). בחרנו בערך 1-, כי בנתוני ציונים זהו ערך בלתי אפשרי, ולכן ניתן להשתמש בו כתנאי גבול.



תרשים 11.8: מערכים של מצביעים למצביעים

מצביעים לפונקציות

השימוש במצביעים לפונקציות אינו שכיח, ופעמים רבות נחליט לכלול אותו בתוכנית רק בדיעבד. כשם שמומלץ להפוך קטעי תוכנית שחוזרים על עצמם לפונקציות, קיימים מקרים בהם נרצה לבצע אותן פעולות עבור טיפוסים שונים של נתונים. במקרים בהם נרצה לבצע פעולות שונות על אותם נתונים נשתמש באפשרות המוצגת כאן. נעיר, ששפת C מאפשרת הגדרת מצביעים לפונקציות, למרות שהפונקציות עצמן אינן משתנות.

נשתמש בדוגמה כדי להסביר את הנושא:

נניח שעלינו לכתוב תוכנית המחשבת היקפים של מעגלים החוסמים ריבועים. התוכנית תקלוט נתונים בשלושה סוגי ייצוג:

א. שתי נקודות בייצוג קרטזי (קואורדינטות x,y).

ב. שתי נקודות בייצוג פולרי (רדיוס וזווית).

ג. אורך צלע.

כדי לפשט את הדוגמה, נציג את ההנחות הבאות:

א. הנתונים ייקראו באופן סדרתי מתוך קובץ נתונים. כל שורה מכילה את אחד משלושת הייצוגים ומסתיימת בתווים '\n' ו-null.

ב. התו הראשון בכל שורה יגדיר את סוג הייצוג: "c", "p", "r" בהתאמה.

ג. הנתונים בקובץ הם בייצוג הקסה דצימלי ואין צורך ב**פעולות הסבה** (conversion) כדי לקבל את ערכם האמיתי. הטיפול המפורט בקבצים יובא בהמשך הספר.

ד. המהדר שבידינו מכיל את הפונקציות הבאות:

sqrt(x) מחזירה $\sqrt{x}$.

sin(x) ,cos(x).

ונניח שקובץ הקלט נראה כך:

```
c 18 02 07 15 \n
p 05 35 06 78 \n
r 07 \n
```

הייצוג בדוגמה הינו בקוד ASCII ומשמש לצורך ההסבר בלבד, כיון שהקובץ מכיל תווים שאינם ניתנים לייצוג.

- **השורה הראשונה מגדירה שתי נקודות בקואורדינטות (7,15) (18,2).**
- **השורה השנייה מגדירה שתי נקודות: ברדיוס 5, זווית 35°.**

  ברדיוס 6, זווית 78°.
- **השורה השלישית מגדירה צלע באורך 7.**

נניח שהפונקציה sqr(x) מוגדרת כך:

```
sqr(x)
   int x;
{
return(x * x);
}
```

כעת נגדיר שלוש פונקציות בעלות ערך יחיד מסוג "מצביע לשלם", אשר מחזירות את האורך של צלע הריבוע.

א. פונקציה לקלט קרטזי:

```
cartezian(inpt_ptr)
  char *inpt_ptr;
{
return (sqrt(sqr(*inpt_ptr - *(inpt_ptr+2)) +
sqr(*(inpt_ptr+1) - *(inpt_ptr+3)))) ;
}
```

ב. פונקציה לקלט פולרי:

```
      /*****     chp1110.c  :מתוך קובץ תקליטור        *****/
polar(char *inpt_ptr)
{
  char otpt_array[4];
                /* הערה: פונקציות טריגונומטריות ב-c מקבלות ערכים ברדיאנים */
otpt_array[0] = *inpt_ptr *
(char)sin(*(inpt_ptr+1) *3.14 / 180) ;
otpt_array[1] = *inpt_ptr *
(char)cos(*(inpt_ptr+1) *3.14 / 180) ;
otpt_array[2] = *(inpt_ptr+2) *
(char)sin(*(inpt_ptr+3) *3.14 / 180) ;
otpt_array[3] = *(inpt_ptr+2) *
(char)cos(*(inpt_ptr+3) *3.14 / 180) ;
return (cartezian(otpt_array));
}
```

ג. פונקציה לקלט אורך צלע:

```
      /*****     chp1110.c  :מתוך קובץ תקליטור        *****/
regular (char *inpt_array)
{
  return (*inpt_array);
}
```

כתוב פונקציה נוספת, לחישוב היקף של מעגל חוסם. הפונקציה תקבל שני ערכים: מצביע לפונקציה המחזירה אורך צלע, ומצביע לשורת הנתונים.

```
           /*****     chp1110.c  :מתוך קובץ תקליטור        *****/

(1)    float calc_circlen(inpt_line, calc_leg)
(2)      char *inpt_line
(3)      int (*calc_leg)() ;
(4)    {
(5)      int leg_len;
(6)      leg_len =(*calc_leg)(inpt_line);
(7)      return(3.14 * sqrt (sqr(leg_len) * 2));
(8)    }
```

הסבר:

א. שים לב להגדרת הערך השני בשורה 3 : ;int (*calc_leg)()

חשוב להקפיד על צורת הגדרה זאת, שהיא ייחודית להגדרת מצביעים לפונקציות. הפונקציה ()calc_leg מוגדרת כאן כפונקציה המחזירה מצביע לשלם.

ב. הקריאה לפונקציה נעשית בהתאם למוסכמה :

הערך המוצבע על ידי ()calc_leg הוא (calc_leg*). זוהי פונקציה, ולכן הקריאה בשורה 6 היא :

```
(*calc_leg)(inpt_line)
```

התוכנית הראשית :

```
        /*****      chp1110.c  :קובץ תקליטור       *****/

(1)    main()
(2)    {
(3)    FILE *f_ptr;
(4)      float len_sum;
(5)      char  input_line[40];
(6)      int   cartezian(char*),polar(char*),regular(char*);

(7)      len_sum=0;
           /* Open input file named "data" */
(8)      f_ptr=fopen("data","r");
(9)      while (fgets(input_line,39,f_ptr) !=0)
           {
(10)       switch (input_line[0])
               {
(11)           case 'c':
(12)            len_sum +=
                   calc_circlen(input_line+1,&cartezian);
(13)            break;
(14)           case 'p':
(15)            len_sum+=calc_circlen(input_line+1,&polar);
(16)            break;
(17)           case 'r':
(18)            len_sum +=
                   calc_circlen(input_line+1,&regular);
(19)            break;
(20)           default:
(21)            printf("\n Non legal input\n");
(22)           }
(23)       }
(24)     printf("Total length of circles is %f\n",len_sum);
(25)   }
```

הסבר:

א. בשורה 6 מוכרזות הפונקציות הייחודיות המשמשות לחישוב צלע הריבוע בכל אחת משיטות הייצוג. הכרזה זו אינה הכרחית, כי ברירת המחדל לטיפוסי פונקציות בשפת C היא שלם (int).

ב. בשורה 8 נפתח קובץ הקלט לקריאה ומתבצעת קריאה סדרתית של שורותיו באמצעות לולאת while שבשורות 9-23.

ג. כל שורת קלט נקראת לתוך מערך הקלט input_line.

ד. באמצעות משפט switch, הבורר לפי תו הקלט הראשון, מתבצעת הקריאה לפונקציית החישוב ()calc_circlen עם המצביע לפונקציה הייחודית המתאימה.

שים לב!

שם הפונקציה (בדומה לשם המערך) הוא מצביע לתחילת הפונקציה.

תרגילים

1. לפניך תמונת זיכרון ותוכנית. מה תהיה תמונת הזיכרון לאחר הרצת התוכנית?

	כתובת	ערך
ptr:	2000	
	2002	
	2004	
	2006	:
	3000	
x:	3002	0
	3004	8
y:	3006	0
	3008	3
	300A	17
	300C	5
	300E	82

```
main()
{
int x=0, y=0, *ptr;

ptr = &y;
*(ptr++) = 8;
*ptr = x;
x = *(ptr + 4);
}
```

2. לפניך מערך שלמים ושני מצביעים. מה יהיו ערכי המערך לאחר הרצת התוכנית?

i_array :	1	2	3	4	5	10	20	30	40	50	60	70	80	90

```
main ()
{
int i_array[] = {1,2,3,4,5,10,20,30,40,50,60,70,80,90};
int *ptr1, *ptr2;
for (ptr1 = ptr2 = i_array; (ptr1-ptr2) < 10; ptr1 += 2)
                  *ptr1 = *ptr2;
*(ptr2 + 12) = 100;
}
```

3. כתוב בעזרת מצביעים את הפונקציה ()str_find, שהוגדרה בשאלה 8 בפרק 6 (מערכים ומחרוזות).

4. כתוב בעזרת מצביעים את הפונקציה ()str_n_cmp, שהוגדרה בשאלה 9 בפרק 6 (מערכים ומחרוזות). כתוב תוכנית קצרה וקריאה.

5. כתוב פונקציה ()str_inv שהופכת מחרוזת ומחזירה מצביע למחרוזת ההפוכה. למשל, עבור המחרוזת "DAVID" הפונקציה תחזיר "DIVAD". מהם הערכים שדרושים לך כדי לבצע פעולה זו?

6. כתוב פונקציה ()str_len_cmp המשווה אורכים של שתי מחרוזות ומחזירה את הערכים הבאים:

❖ ערך שלילי אם המחרוזת הראשונה קצרה מהשנייה.

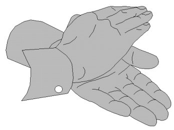
❖ ערך 0 (אפס) אם אורכיהן שווים.

❖ ערך חיובי אם המחרוזת השנייה קצרה מהראשונה.

❖ אין להשתמש בפונקציה ()strlen, ויש לכתוב תוכנית יעילה וקצרה.

7. נתון מערך של מצביעים למחרוזות. כתוב פונקציה, אשר מחפשת את המופעים של תו כלשהו במערך הנתון. הפונקציה מקבלת את המערך באמצעות מצביע ואת התו לחיפוש. היא מחזירה את מספר המופעים של התו הזה בכל המחרוזות המוצבעות על ידי המערך.

8. כתוב תוכנית ראשית ופונקציה. הפונקציה מקבלת כערך את המערך שבשאלה 7, ומחזירה את מספר המופעים של התו השכיח ביותר במחרוזות המוצבעות על ידי המערך. הפונקציה תציב את התו הזה במשתנה max_char (מטיפוס תו), השייך לתוכנית ()main. התוכנית הראשית תדפיס אותו ואת מספר המופעים שלו במחרוזות. בחר ערכים מתאימים וקרא לפונקציה באופן נכון.



משימות:

תרגילים 13-33.

פרק 12

מבנים

מבנה (Structure) הינו אוסף של משתנים מטיפוס זהה או שונה, המקובצים יחד תחת שם אחד. השימוש במבנה מקל על הטיפול בכמה משתנים בעלי קירבה בתוכנם, או במידע אותו הם מכילים. האפשרות לצרף כמה משתנים במבנה משולב, מקלה על הגדרת מבני נתונים מורכבים ומאפשרת גישה מהירה ויעילה לנתונים שבתוכם.

תיאור והגדרה של מבנים

הגדרת מבנה:

```
struct <שם סוג המבנה>
{
  :
  :
  :   הגדרת שדות בעזרת טיפוסים בסיסיים - תבנית
  :
  :
} < רשימת משתני מבנה >;
```

תרשים 12.1: תיאור סכימטי של מבנה

הסבר:

א. מילת המפתח השמורה **struct** פותחת הגדרה של **מבנה**, ועליה להופיע בתחילת ההגדרה.

ב. **<שם סוג המבנה>**: הגדרה זו הינה רשות. אם היא מופיעה - היא תוצמד להגדרת המבנה, כך שניתן יהיה להתייחס למבנה זה בעתיד באמצעות שם זה. מנקודת ההגדרה הזו תכיר המערכת טיפוס חדש (בנוסף לטיפוסים הבסיסיים שכבר הכרנו) מסוג:

struct <שם סוג המבנה>

ג. **הגדרת שדות**: את תכולת המבנה מגדירים בתוך סוגריים מסולסלים באמצעות טיפוסים בסיסיים המוכרים למערכת. למכלול ההגדרה הזו אנו מכנים **תבנית**. התבנית מקנה את צורתו ותכולתו של המבנה.

ד. <**רשימת משתני מבנה**>: הגדרה זו הינה רשות. המשתנים המופיעים ברשימה, במידה והיא קיימת, יהיו משתני מבנה בעלי תבנית זהה להגדרה. אם לא מציינים רשימה כזו, ישמש משפט ההגדרה להצגת טיפוס חדש שנשתמש בו בהמשך.

נקל לראות, שהגדרת מבנה ללא <שם סוג המבנה> וללא <רשימת משתני מבנה> היא חסרת משמעות, כי אין כל אפשרות להשתמש בה.

ה. לאחר שמגדירים מבנה מטיפוס חדש, ניתן להגדיר בצורה מקוצרת משתני מבנה בעלי תבנית זהה, בעזרת ההגדרה הבאה:

```
struct <שם סוג המבנה> <רשימת משתני מבנה>;
```

<שם סוג המבנה> **חייב** להופיע כאן, ובלעדיו אין להגדרה כל משמעות.

נשתמש בדוגמת יישום כדי להציג את אפשרויות השימוש במבנים.

נניח שברצוננו לתאר פריט, או מוצר מסוים, במסגרת תוכנית מחשב המסייעת בניהול חנות כל-בו. לכל מוצר יש מספר מאפיינים: קוד-מוצר, שם-מוצר, מחלקה, מחיר, כמות הנמצאת במלאי, כמות מינימום במלאי ושם ספק המוצר. את המאפיינים האלה אפשר לכלול במבנה, כדוגמת רשומה המכילה שדות של נתונים שונים:

```
struct product  {
                  int  pr_code;
                  char  pr_name[7], department[10];
                  float price;
                  int  inventory, min_inv ;
                  char  supplier[10];
                } prod;
```

הסבר:

א. בדוגמה זו, <שם סוג המבנה> הוא product. השם product קשור לתיאור אופי הנתונים המוכלים במבנה, אשר מפורט בין הסוגריים המסולסלים.

ב. בתוך הסוגריים המסולסלים מוגדרים משתנים מטיפוסים אשר מוכרים לנו: float, int ו-char. הם מהווים את תבנית ההגדרה של המבנה product.

ג. שם המבנה שבדוגמה - prod - הוא משתנה מטיפוס struct product.

נציג באופן סכימטי את המבנה (או הרשומה) שבדוגמה:

prod

int	char	char	float	int	int	char
pr_code	pr_name[7]	department[10]	price	inventory	min_inv	supplier[10]

תרשים 12.2: מבנה של רשומה לדוגמה

כל אחד מהמשתנים המוגדרים בתבנית המבנה, נקרא **שדה** (Field). על מנת לפנות לשדה מסוים נכתוב:

prod.<שם השדה>

שים לב ל**נקודה** (.) **המחברת** את שם השדה אל שם המבנה, כמו למשל:

prod.pr_code קוד המוצר של המבנה prod.

prod.price המחיר של המוצר במבנה prod.

אילו הסתפקנו בהגדרת מוצר בודד, כדוגמת prod, לא היה יתרון מיוחד "לארוז" במבנה את המשתנים המתארים מוצר. אפשר היה להגדיר את המשתנים באופן רגיל ולוותר על הקידומת prod. יעילות השימוש במבנים באה לידי ביטוי כאשר מנצלים אותם כדי להגדיר מספר רב של משתני-מבנה.

המשפט הבא מגדיר 3 משתנים מסוג מבנה-מוצר:

```
struct product p1,p2,p3
```

כל אחד מהמשתנים p_i מכיל את אוסף השדות שהוגדר קודם, בתבנית ההגדרה product.

כדי להדפיס קטלוג מחירים של שלושת המוצרים, אפשר לכתוב את הפקודות הבאות:

```
printf ("\t product \t\t\t price \n");
printf ("\t %s \t\t\t %5.2f \n",p1.pr_name,p1.price);
printf ("\t %s \t\t\t %5.2f \n",p2.pr_name,p2.price);
printf ("\t %s \t\t\t %5.2f \n",p3.pr_name,p3.price);
```

הצירוף "struct product" במשפט ההגדרה ממלא תפקיד של "סוג טיפוס משתנה" במשפטי ההגדרה שהכרנו עד כה. המשפט שהוגדר מקביל בתפקידו להגדרה:

```
int i1,i2,i3;
```

באמצעות הגדרת המבנה product הוספנו לארבעת הטיפוסים שהכרנו טיפוס נוסף המותאם לדרישה מסוימת - הגדרת מבנה. נחזור לדוגמה זו בהמשך.

דוגמה נוספת:

ברצוננו לתאר "זמן" על ידי הצגת שעות, דקות ושניות. לשם כך נגדיר טיפוס חדש באמצעות מבנה מתאים.

```
struct time  {
         int hour
         int min
         int sec
             }  ;
```

הגדרה זו שקולה להגדרה המקוצרת הבאה:

```
struct time {
         int  hour, min, sec;
            } ;
```

מכיון שהשמטנו את <רשימת המשתנים>, משפט זה מגדיר תבנית בשם time. בשלב זה עדיין לא הגדרנו משתנים כלשהם. בזמן ההידור, המהדר שומר את התבנית ומסמן לעצמו טיפוס חדש מסוג struct time. כאשר נציין רשימת משתנים צמודה לטיפוס זה, תתבצע הגדרת משתנים ממשית ויוקצה להם מקום בזיכרון.

במשפט ההגדרה הבא יהיו t1, t2 ו-t3 מבנים מטיפוס time.

```
struct time t1,t2,t3;
```

בהנחה שהמשתנים t_i מכילים כל-אחד נתונים של זמן, ניתן לכתוב קטע תוכנית שיציב ל-t3 את הפרש הזמנים בין t1 ל-t2.

```
      /*****     chp1201.c  :קובץ תקליטור      *****/

      #include <stdio.h>

(1)   struct  time {
               int  hour;
               int  min;
               int  sec;
                  };

(2)   struct  time t1={10,20,5}, t2={14,50,25}, t3;

(3)   void main(void)
(4)   {
(5)     t3.hour = t1.hour - t2.hour;
(6)     t3.min = t1.min - t2.min;
(7)     t3.sec = t1.sec - t2.sec;
(8)     if (t3.sec < 0)
          {
(9)     t3.sec += 60;
(10)      --t3.min;
(11)      }

(12)    if (t3.min < 0)
          {
(13)      t3.min += 60;
(14)      --t3.hour;
(15)      }
(16)    printf("The time difference is \n");
(17)    printf("%d hours, %d minutes, %d seconds\n",
                                 t3.hour, t3.min, t3.sec);
(18)  }
```

במשפטי if שבשורות 8 ו-12 אנו מעדכנים שדה בעל ערך שלילי על ידי תהליך "השאלה" משדה משמעותי יותר. לדוגמה, אם שדה השניות שלילי - נוסיף לו 60 שניות ונחסר דקה אחת משדה הדקות. תהליך דומה נעשה בין שדה הדקות לשעות.

אתחול מבנים

בדוגמה הקודמת אנו מסתמכים על כך, שב-t1 וב-t2 קיימים כבר ערכים. מתוכן התוכנית ניתן לראות שערכים אלה לא הוצבו עדיין למשתנים, ועלינו ללמוד איך עושים זאת.

קיימות שתי שיטות להכנסת נתונים למבנה:

א. באמצעות פקודות הצבה.

ב. באמצעות אתחול.

אם ברצוננו להציב ל-t1 את השעה 13:33:20 (שעה 13, 33 דקות ועשרים שניות), נוכל לעשות זאת כך:

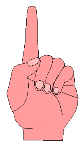
```
struct time t1;
t1.hour = 13;
t1.min = 33;
t1.sec = 20;
```

בדומה לאתחול המערך, אפשר לכתוב גם כך:

```
struct time t1 = {13,33,20};
```

ההגדרה הכללית היא:

struct <שם סוג המבנה> <שם משתנה> **=** **{**<רשימת ערכי אתחול>**}**;



שים לב!

<רשימת ערכי אתחול> נמצאת בין סוגריים מסולסלים. שדות המבנה יאותחלו לערכי הרשימה בהתאם לסדר הגדרתם בתבנית. אם <רשימת ערכי האתחול> מכילה פחות ערכים ממספר שדות המבנה, יאותחלו השדות הנותרים לאפס. חוק זה ישים גם כשאין אתחול כלל! כלומר, במבנה חדש (סטטי או גלובלי) יאותחלו כל השדות לאפס, אם לא הוגדר אחרת.

אם בשורה (1) של התוכנית הקודמת נכתוב:

```
(1)  struct time t1={10,20,5}, t2={14,50,25} ,t3;
```

יהיה הפלט בסיום ריצת התוכנית:

```
The time difference is:
-5 hours, 29 minutes, 40 seconds
```

שים לב!

הערך השלילי מתייחס אך ורק לשדה השעות. כלומר: 5- שעות ועוד 29 דקות ועוד 40 שניות.

נסה לשנות את הערכים שבשורת האתחול והרץ את התוכנית. בחן את התוצאות בהתאם לערכים שהצבת.

כדי לקבל הפרש זמן בערך מוחלט, נוסיף מספר פקודות בסוף קטע התוכנית הקודם, אחרי שורה 15:

```
       /*****      מתוך קובץ תקליטור: chp1202.c      *****/

   if (t3.hour < 0)
      {
      t3.hour = abs(t3.hour+1);
      t3.min = 59-t3.min;
      t3.sec = 60-t3.sec;
      }
   printf("The time difference is \n");
   printf("%d hours, %d minutes, %d seconds\n",
                   t3.hour, t3.min, t3.sec);
}
```

כעת, הפלט בסיום התוכנית יהיה:

```
The time difference is:
4 hours, 30 minutes, 20 seconds
```

מצביעים למבנים

כאשר עסקנו במערכים ראינו שניתן להתייחס למערך בשלמותו על ידי שימוש בכתובת שלו. כך גם במבנים - ניתן לקבל את כתובת המבנה ולהשתמש בערך זה בתוכנית.

מכיון שמבנה יכול להכיל מספר רב של איברים, בדומה למערך, העברת מבנה בשלמותו לפונקציה היא תהליך מאוד לא יעיל. שפת C מבצעת, כזכור, העברה "לפי-ערך" ולכן תתבצע העתקה של כל ערכי המבנה לתוך מבנה זמני וממנו למבנה היעד. פתרון טוב יותר יהיה להעביר את **כתובת המבנה** למשתנה פנימי של פונקציה, שיוגדר לצורך כך **כמצביע למבנה**. צורת ההגדרה הכללית תהיה:

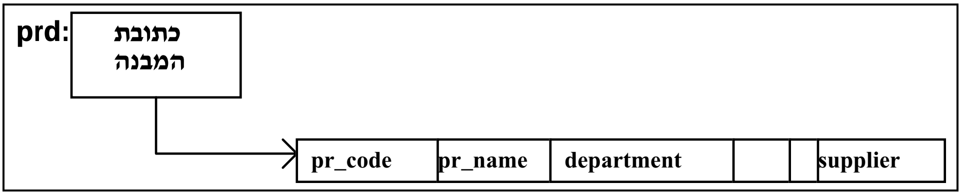
```
struct <שם סוג מבנה> * <שם מצביע>;
```

לדוגמה:

```
struct  product *prd;
```

prd הינו **מצביע** למבנה מסוג product.

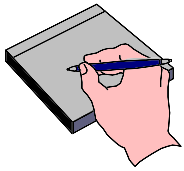
הסכימה הבאה מתארת את קשרי הגומלין בין המצביע לבין המבנה:



תרשים 12.3: מצביע למבנה

כאשר פונים למשתנה המוצבע על ידי מצביע, מוסיפים כוכבית (*) לפני שם המצביע. למשל, כדי לפנות למחיר המוצר price, המוצבע על ידי המצביע prd, נכתוב:

```
(*prd).price
```



הערה!

כאשר משתמשים במצביע למבנה, חשוב להקפיד שהמצביע יוגדר מטיפוס מצביע.

פנייה מהסוג: (*prd).hour אינה חוקית, מכיון ש-prd מצביע למבנה מסוג product ולא למבנה מסוג time. כדי למנוע טעויות מסוג זה, מומלץ לכלול בשם המצביע חלק משם המבנה, וליצור קשר סמנטי כלשהו ביניהם. למשל:

```
*tim_ptr
```

או

```
*prd_ptr
```

עקב השימוש הנפוץ במבנים, קיימת בשפת C דרך נוספת לפנות למשתנה במבנה. אפשר לכתוב:

```
(*prd).price
```

או כך:

```
prd->price
```

סימן "החץ" (מקף וסימן גדול מ-) מיועד אך ורק עבור מצביעים למבנים ולאיגודים (אותם נכיר בפרק הבא).

מצביעים למבנים כערכים לפונקציות

משתמשים במצביעים למבנים, כדי להעביר בעזרתם מבנה שלם כערך לפונקציה. באמצעות פונקציה ניתן להפעיל את קטע התוכנית על מבנים רבים ולבצע בהם פעולות רבות.

נציג עתה סדרת פונקציות על בסיס המבנה product. את הפונקציות נקשור יחד, באמצעות תוכנית ראשית ()main ונקבל "חבילת תוכנה" המיועדת לניהול חנות כל-בו. השלבים שיוצגו הם שלבים בסיסיים בתכנון ופיתוח של מערך תוכנה, ומוצע לנהוג לפיהם במיוחד כאשר בונים תוכניות גדולות ומורכבות.

תכנון ראשוני, הגדרת מטרות התוכנית

א. עדכון מחיר מוצר.

ב. עדכון כמות המוצרים במלאי בשני מקרים:

❖ כשמגיעה אספקה חדשה לחנות.

❖ בסוף היום, לאחר שנמכרו מוצרים.

ג. הדפסה של רשימת כל המוצרים בחנות.

ד. הדפסה חלקית של רשימת המוצרים, שיש להזמינם מהספק. התנאי לכך הוא, שהכמות במלאי קטנה מהמינימום שנקבע עבור אותו מוצר.

ה. חישוב הערך הכספי של הסחורה המצויה בחנות, על פי הערך של כל מוצר.

בשלב זה נסתפק במטרות אלו, ונכתוב עבור כל משימה פונקציה נפרדת. אחר-כך נחבר את כל הפונקציות לחבילת תוכנה אחת, שאותה נוכל להפעיל ישירות ממסוף המחשב לשם ניהול החנות.

פונקציות שירות בסיסיות

א. עדכון מחיר של מוצר:

```
       /*****      chp1203.c  :מתוך קובץ תקליטור        *****/
void updtprice (struct product *prd, float newprice)
{
  prd->price = newprice;
}
```

ב. עדכון כמות המוצרים במלאי:

```
       /*****      chp1203.c  :מתוך קובץ תקליטור        *****/
void updtquant (struct product *prd, int q)
{
  prd->inventory += q;
}
```

q הוא ערך המכיל את מספר הפריטים שהתקבלו (חיובי), או כמות הפריטים שנמכרו (שלילי).

ג. תדפיס של מוצר בודד:

```
       /*****      chp1203.c  :מתוך קובץ תקליטור        *****/

void printprod (struct product *prd)
{
  printf("%4d %7s %10s %5.2f %9d %10s\n",
     prd->pr_code, prd->pr_name, prd->department,
        prd->price, prd->inventory, prd->supplier);
}
```

ד. חישוב הערך הכספי של המלאי של כל מוצר בחנות:

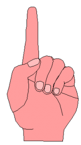
```
      /*****     chp1203.c  :מתוך קובץ תקליטור      *****/

float money(struct product *prd)
{
  return(prd->inventory * prd->price);
}
```

ה. הדפסת כותרת לדוח מוצרים:

```
      /*****     chp1203.c  :מתוך קובץ תקליטור      *****/

void headline(void)
{
  printf("Code_
         Product Department Price Inventory Supplier\n");
}
```



שים לב!

את משפט printf() יש לכתוב ברצף כמשפט אחד ארוך, ואז אין צורך לכתוב את המקף התחתון אלא תו רווח אחד, או יותר כנדרש.

תוכנית לבדיקת פונקציות השירות

נכלול את הפונקציות הבודדות בחבילה אחת ונדגים את פעולתן. את חבילת התוכנה לניהול החנות נבנה בהמשך, לאחר שנלמד מספר נושאים נוספים.

```
      /*****     chp1203.c  :מתוך קובץ תקליטור      *****/
struct product pr = {
  155, "MILK", "FOOD", 2.46,100,50, "TNUVA"
     };
void main(void)
{
  headline();
  printprod(&pr);
  printf("Change price to 2.83 and sell 50 items\n");
  updtprice(&pr, 2.83);
  updtquant(&pr, -50);
  headline();
  printprod(&pr);

  printf("How much is the store inventory worth?\n");
  printf("Inventory of %s is %f shekel.",
                 pr.pr_name,money(&pr));
}
```

תוצאות הרצת התוכנית:

```
Code   Product Department  Price Inventory Supplier
155    MILK    FOOD        2.36  100       TNUVA

Change price to 2.83 and sell 50 items.

Code   Product Department Price Inventory Supplier
155    MILK    FOOD       2.83  50        TNUVA

How much is the store inventory worth?
Inventory of MILK is 141.5 shekel.
```

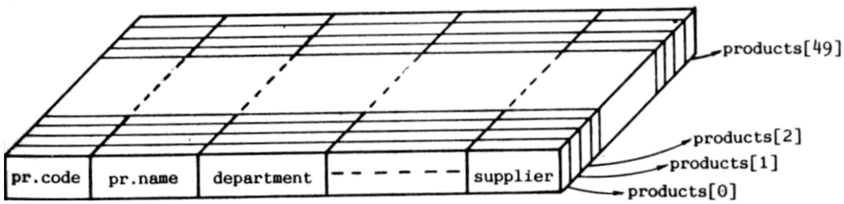
מערכים של מבנים

התוכנית שכתבנו עבור חנות הכל-בו עדיין אינה שלמה, כי הפונקציות שכתבנו מטפלות במוצר אחד מסוים, בשעה שבחנות כל-בו ממוצעת קיימים מאות מוצרים. אם נשתמש בשיטת ההגדרה היחידנית, יהיה עלינו להגדיר כל אחד מהם בנפרד, מלאכה שאינה אפשרית בפועל.

נוכל לפתור קושי זה על ידי מערך של מבני מוצרים. הטיפוס struct product שהגדרנו, זהה בתכונותיו לטיפוסים הבסיסיים. עתה, נוכל להגדיר באמצעותו **מערך של מבנים** מטיפוס זה. נניח שבחנות שלנו יש חמישים מוצרים שונים, ונקבל:

```
struct product products[50];
```

זהו מערך בן 50 מבנים שכל אחד מהם הוא מטיפוס struct product. באופן סכימטי אפשר להציג את המערך product בתרשים 12.4.



תרשים 12.4: *מערך של מבנים*

כדי לפנות למחיר של המוצר החמישי במערך, נכתוב:

```
products[4].price
```

באופן דומה, הספק של המוצר העשרים יהיה:

```
products[19].supplier
```

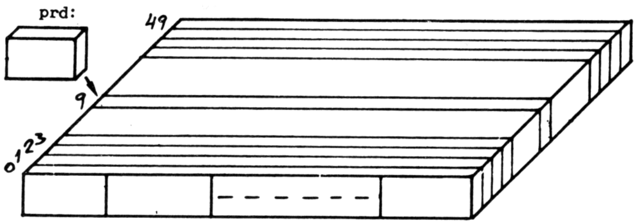
באופן כללי, נפנה כך לשדה כלשהו של המוצר ה-i :

products[i-1].< שם השדה>

בטיפול במערכים של מבנים מתגלה שימוש נוסף למצביעים למבנים, אשר מקל את הגישה למבנים במערך ומקנה יכולת נוחה לבצע פעולות במערך.

```
(1)  struct product *prd;
(2)  prd = &products[9];
```

פקודה 2 מציבה למצביע prd את הכתובת של המבנה העשירי במערך. באופן סכימטי מוצגת אפשרות זו בתרשים 12.5.



תרשים 12.5: מצביע אל מערך כמערך של מבנים

באמצעות prd ניתן לפנות לכל אחד משדות המבנה products[9]. כדי לגשת לשדה כלשהו נכתוב :

prd-><שם שדה>

או באופן מפורש :

```
prd->pr_name, prd->price.
```

מכיון ש-prd הוגדר כמצביע למבנה מטיפוס product, הרי שהפקודה ;prd++ תקדם את prd, כך שיצביע על **המבנה הבא**. הערך (או הכתובת) הנמצא ב-prd יקודם **בגודל של המבנה product**. המהדר "אחראי" לעקוב עבורנו אחר העובדה ש-prd מצביע למבנה מטיפוס struct product, ולכן "ידע" לתרגם את פקודת הקידום כראוי.

הפקודות הבאות תבצענה, גם הן, את אותה פעולה :

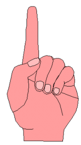
```
prd = prd + 1;
prd += 1;
```

גודל המבנה הינו סכום גודלי השדות המרכיבים אותו. במערכת מחשב, בה הגדלים של הטיפוסים הבסיסיים הינם char - בית אחד, int - 2 בתים, float - 4 בתים, גודלו של מבנה מטיפוס product יהיה 37 בתים. שפת C מאפשרת להתייחס לגודל זה ללא צורך בחישוב כלשהו, או ידיעת הגדלים של הטיפוסים הבסיסיים, באמצעות **הפקודה sizeof** ("גודל של"). זוהי **פקודת "פסידו"**, אשר מבוצעת בזמן ההידור. ניתן להציב את הערך המתקבל למשתנה.

נגדיר:
```
int prod_siz ;
```
ונכתוב כך:
```
prod_siz = sizeof (struct product);
```
או כך:
```
prod_siz = sizeof (prd);
```
כל אחת מהפקודות האלו תבוצע על ידי המהדר בעת ההידור, תחשב את גודל המבנה ותציב אותו כערך מוחלט. בדוגמה זו נקבל:
```
prod_siz = 37;
```
הערך המועבר ל-sizeof הוא טיפוס, או מצביע לאותו טיפוס. נציין שוב שפעולת sizeof מבוצעת בזמן ההידור ואין לה כל השפעה בזמן ריצה. לכן, פעולות הקידום של המצביע prd מוסיפות לערכו את הערך sizeof (struct product). פעולת קידום של מצביע, שאינו מצביע למבנה כלשהו מתוך מערך היא חסרת משמעות.

נציין, שאופרטור החץ (<-) הקשור למצביע למבנה, חזק יותר מאופרטור הקידום ++. לכן פקודה כגון: ++prd>inventory- תקדם את **ערך המלאי** של המבנה המוצבע על ידי prd, ולא תפנה לשדה המלאי של המבנה הבא! כדי לבצע פנייה כזו נשתמש בסוגריים, שלהם דרגת הקדימות הגבוהה ביותר: ++prd)->inventory).

הפקודה:
```
prd++->inventory;
```
תפנה לשדה המלאי של המבנה המוצבע על ידי prd, ולאחר מכן תקדם את prd למבנה הבא.

פקודה זו שקולה לפקודה:
```
(prd++)->inventory;
```


שים לב!

הסוגריים מיותרים. מדוע?

נותרה עדיין השאלה כיצד נאתר מוצר בודד מתוך מערך המוצרים?

מכיון שבנינו פונקציות המטפלות במוצר בודד, עלינו לפתח דרך נוחה לאתר מוצר מסוים "ולשלוח" אותו לאחת מהן. בדרך כלל נוח לבצע איתור כזה על פי המספר הקטלוגי ("קוד") של המוצר. שיטה זו עדיפה על פני איתור המוצר על פי שמו, מכיון ששגיאות כתיב נפוצות ועלולות לגרום לאיתור שגוי (או לחוסר איתור).

המוצרים שבחנות הכל-בו מסודרים במערך של מבנים מטיפוס struct product. נכתוב פונקציה המאתרת מבנה של מוצר על פי מספר הקוד שלו ומחזירה מצביע למבנה שאותר. את המצביע הזה ניתן להעביר לכל אחת מהפונקציות שכבר כתבנו, ולבצע עליו את הפעולה הנדרשת.

פונקציה לאיתור מוצר

```
       /*****      chp1204.c  :מתוך קובץ תקליטור      *****/
(1)  struct product *findprod(struct product prods[],
                              int prdcode)
(2)
(3)  {
(4)      struct product *p;
(5)      for (p=prods ; p->pr_code != prdcode : ++p)
              ;
(6)      return(p);
(7)  }
```

הסבר:

א. הפונקציה ()findprod הוגדרה מטיפוס מצביע ל-struct product. הגדרה זו תואמת את כל העובדות שהכרנו עד כה, בדבר טיפוסי פונקציות וערכים מוחזרים.

ב. גוף הפונקציה מורכב מפקודת for ריקה (שורה 5) הסורקת את מערך המוצרים, על ידי קידום המצביע p. הסריקה תתבצע עד שנמצא שוויון בין קוד המוצר המבוקש לבין קוד המוצר המוצבע על ידי p, או עד שמגיעים לסוף המערך. כאן אנו מניחים שערך שדה הקוד של המוצר האחרון הוא אפס.

אתחול מערך של מבנים

אתחול מערך מבנים נעשה בדרך דומה לאתחול של מערך רגיל. נכתוב בצורה כללית:

```
struct <שם סוג מבנה> <שם מבנה>[] = {
      { <ערך שדה 1>, <ערך שדה 2>,  ...},
      { <ערך שדה 1>, <ערך שדה 2>,  ...},
           ...
      { <ערך שדה 1>, <ערך שדה 2>,  ...},
            } ;
```

נאתחל לדוגמה, את מערך המבנים []products של חנות הכל-בו:

```
       /*****      chp1204.c  :מתוך קובץ תקליטור      *****/
struct product products[50] = {
  {123,"CHEEZE","FOOD",3.54,30,20,"TNUVA"},
  {155,"MILK","FOOD",2.46,100,50,"TNUVA"},
  {147,"YOGART","FOOD",3.50,50,20,"TENE"},
  {133,"CORNFLAKES","FOOD",5.60,30,10,"TELMA"},
  {151,"MARGARINE","FOOD",1.20,70,50,"TELMA"},
  {170,"BREAD","FOOD",0.70,200,100,"KIMEL"},
  {230,"LEVIS","CLOTHES",25.0,30,20,"ATA"},
  {251,"LEVIS","CLOTHES",52.0,30,20,"ATA"},
  {262,"NIKE","SHOES",120.0,40,20,"NIKE"},
  {221,"SNEAKERS","SHOES",75.0,50,30,"GALI"},
  {450,"NOTEBOOK","OFFICE",4.50,100,20,"DEFRON"},
  {0,"","",0.0,0,0,""}
     };
```

הסבר:

א. אתחול של כל מבנה מבוצע באמצעות רשימת ערכים לשדות המבנה, כשהללו מוקפים בזוג סוגריים מסולסלים שאחריהם פסיק.

ב. אתחול המערך מבוצע באמצעות רשימת אתחולים לכל מבנה במערך כשהללו מוקפים בזוג סוגריים מסולסלים ופסיק אחריהם. זוהי דוגמה נוספת לאחידות בשפה.

ג. המבנה האחרון מכיל אפסים ומטרתו לסמן את סוף מערך המבנים.

בהסתמך על דוגמת האתחול, נדגים את השימוש בפונקצית האיתור שכתבנו:

```
      /*****      chp1204.c :מתוך קובץ תקליטור       *****/

void main(void)
{
struct product *prd;
printf("Locate product with code 170\n");
prd = findprod(products,170);
headline()
printprod(prd);
}
```

הפלט מהרצת התוכנית:

```
Locate product with  code 170
Code Product Dept.   Pric. Inventory Supplier
170  BREAD   FOOD    0.70  200       KIMEL
```

המצביע prd, שהשתמשנו בו בתוכנית הקודמת - מיותר כעת, כי השימוש היחיד בו נעשה לצורך קליטת התוצאה של פונקצית האיתור ()findprod והעברתה לפונקציה ()printprod. אפשר לוותר על איתור מראש של המוצר לשם הדפסה, כי ניתן להפעיל את הפונקציה ()findprod ישירות כערך של הפונקציה ()printprod. דוגמה לכך ניתנת בתוכנית הבאה:

```
(1)  void main(void)
(2)  {
(3)  struct product *findprod();

(4)  printf (locate product with code 170\n");
(5)  headline();
(6)  printprod (findprod (170));
(7)  }
```

חבילת תוכנה לניהול בית כל-בו

בסעיפים הקודמים הצגנו את מהלך התכנון וחלקים שונים של חבילת התוכנה לניהול בית כל-בו. עתה נוכל להדגים את התוכנה בשלמותה. מטרתנו להפעיל את התוכנה ממסוף, ועל-כן נקבע סדרת קודים, אשר נקיש במקלדת לביצוע הפעולות השונות:

- **P** - עדכון מחיר מוצר.
- **S** - עדכון כמות שנמכרה.
- **I** - עדכון כמות שהתקבלה.
- **M** - חישוב והדפסת הערך הכללי של הסחורה בחנות.
- **V** - חישוב והדפסת הפדיון היומי של החנות.
- **T** - הכנת תדפיס כללי של כל המוצרים בחנות.
- **R** - הכנת תדפיס של כל המוצרים הנמצאים מתחת למינימום.
- **E** - סיום עבודה עם החבילה.

כדי לבצע פעולה כלשהי נקיש את הקוד של סוג הפעולה הנדרשת. התוכנית תקלוט קוד זה, תבדוק אם הוא E, כלומר "סוף", ואם לא - היא תבצע את הפעולה. לכל קוד פעולה יש קטע תוכנית עצמאי, אשר נפנה אליו באמצעות משפט הבקרה switch. באופן סכימטי נציג זאת כך:

```
void main(void)
{
   אתחול מערך המוצרים;
   while (קליטת קוד הפעולה ! = 'E')
      בצע את הפעולה לפי הקוד שהוקלד;
}
```

התוכנית הראשית:

```
      /*****      chp1205.c  מתוך קובץ תקליטור:       *****/

struct product products[50] = {
                        {          הנתונים מופיעים }
                        {            בקובץ התקליטור }
                        {      0,"","",0.0,0,0,"" }
                          } ;
  ....
void main(void)
{
   struct product *prd;
   float total, value, newtot;
   char code;
```

```
  while ((code=getcode()) !='E')
  {
    switch(code)
        {
      case 'P':
         newprice();
         break;
      case 'S':
         sell();
         break;
      case 'I':
         supply();
         break;
      case 'M':
         for(total=0,prd=products;prd->pr_code;++prd)
         total += money(prd)
    printf("The value of all products in the store is
                  %f\n", total);
         break;
      case 'V':
         for(newtot=0,prd=products;prd->pr_code;++prd)
         newtot += money(prd);
         value = total - newtot;
    printf("Total value of products sold today:
                  %f Shekel\n", value);
         break;
      case 'T':
         headline();
         for(prd=productes;prd->pr_code;++prd)
            printprod(prd);
         break;
      case 'R':
         headline();
         for(prd=products;prd->pr_code;++prd)
            if(prd->inventory < prd->min_inv)
                   printprod(prd);
        }
  }
    printf("Program ended\n");
}
```

הסבר:

א. את הטיפול בשלושת סוגי הפעולה הראשונים לא פירטנו, כיון שיש צורך להשתמש בפונקציות מורכבות שעדיין לא למדנו. אנו ממליצים להשאיר את התוכנית הראשית כמסגרת, ולא לכלול בה קטעים ארוכים ומורכבים. התוכנית קריאה ומובנת יותר, כאשר מוצג בה הרצף הלוגי של הפעילויות והטיפול בפרטים חבוי בתוך הפונקציות.

ב. מטרת הפונקציה ()getcode, שמופעלת בשורות הראשונות של התוכנית, היא להחזיר תו שהוקלד במקלדת. בהמשך נראה את הביצוע, כעת נסתפק בהנחה שהקוד שתקבל יהיה חוקי (היינו, אחד מאלה שמותרים בתוכנית).

ג. בארבעת קודי הפעולה הנותרים מתבצע מעבר סדרתי על כל המוצרים בחנות. המעבר מתבצע באמצעות לולאת for, המקדמת מצביע על פני המערך כולו.

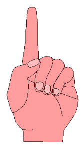
כעת נכתוב את הפונקציות החסרות.

עדכון מחיר של מוצר

```
     /*****      chp1205.c  :מתוך קובץ תקליטור      *****/

(1)   void newprice(void)
(2)   {
(3)      int pcode;
(4)      float pric;
(5)      struct product *p;

(6)      printf("Enter product code:");
(7)      scanf("%d", &pcode);
(8)      p = findprod(products, pcode);
(9)      printf("Old price is %s is %f Shekel\n",
                                p->pr_name, p->price);
(10)     printf("Enter new price:");
(11)     scanf("%f", &pric);
(12)     updtprice(p, pric);
(13)  }
```



שים לב!

אנו משתמשים בפונקציה ()scanf בשורות 7, 11. קליטת הנתונים נעשית בעזרת כתובת המשתנים, כפי שלמדנו. העברת המשתנה עצמו הינה טעות נפוצה בתוכניות ואם נעשה זאת - הערכים לא יגיעו ליעדם.

עדכון כמות המלאי

```
    /*****      chp1205.c  :מתוך קובץ תקליטור       *****/

(1)   void sell(void)
(2)   {
(3)     int pcode, quant;

(4)     printf("Enter product code:");
(5)     scanf("%d", &pcode);
(6)     printf("Enter number of items sold:");
(7)     scanf("%d", &quant);
(8)     updtquant(findprod(products,pcode), -quant);
(9)   }
```

לאחר שקלטנו את קוד המוצר (בשורה 5) ומספר היחידות שנמכרו ממנו, אנו קוראים ל-()updtquant. אחד מערכיה הוא פונקצית האיתור (המחזירה מצביע, כנדרש) והערך השני הוא הכמות (בערך שלילי, כי זוהי הפחתה מהמלאי).

עדכון כמות שהתקבלה

```
      /*****      chp1205.c  :מתוך קובץ תקליטור       *****/
void supply(void)
{
  int pcode, quant;
  printf("Enter product code:");
  scanf("%d", &pcode);
  printf("Enter number of items brought:");
  scanf("%d", &quant);
  updtquant(findprod(products,pcode), quant);
}
```

בדיקת חוקיות של תו "סוג פעולה"

```
     /*****      chp1205.c  :מתוך קובץ תקליטור       *****/
 (1)  char  getcode()
 (2)  {
 (3)     char  c, ok;
 (4)     do {
 (5)     c = getch();
 (6)     ok=(c=='P') || (c=='S') || (c=='I') || (c=='M') ||
 (7)        (c=='V') || (c=='T') || (c=='R') || (c=='E') ;
 (8)     if (!ok)
 (9)       printf("%c is an unknown code, try again\n",c);
(10)     }
```

```
(11)        while (!ok);
(12)        return(c);
(13)    }
```

הסבר:

א. לולאת do-while מתבצעת כל עוד המשתנה ok אינו אמת.

ב. מתבצעת בדיקת חוקיות הקוד. ברגע שנמצא תו - הבדיקה נפסקת, אחרת - מתבצעת בדיקה מלאה. זוהי דרך מקובלת לבדוק תנאים לוגיים.

ג. התו המוחזר בשורה 12 מכיל קוד חוקי בלבד, כנדרש.

מבנים בתוך מבנים - דוגמת יישום

עד עתה ניצלנו את העובדה שמבנה הוא טיפוס, כדי שנוכל להגדיר מערכים של טיפוס זה. נתקדם צעד נוסף ונגדיר את אחד משדותיו של מבנה מסוים כ**שדה מסוג מבנה**.

תוכנית לרישום נוכחות וחישובי שכר

בדוגמה הבאה מוצגת תוכנית מחשב המשמשת לרישום נוכחות במפעל ולחישוב שכר. בתוכנית זו, לכל עובד יהיה מערך של 25 מבנים, אחד לכל יום עבודה בחודש. כל מבנה יכיל את המידע הבא: שם העובד, שעת כניסה למפעל ושעת יציאה. בסוף החודש ניתן יהיה לחשב באמצעות סריקת המבנים, את סה"כ שעות העבודה של אותו עובד בחודש ואת מספר השעות הנוספות (מעל 8 שעות נוכחות ביום). באמצעות המידע הזה נחשב את המשכורת החודשית של העובד, וניקח בחשבון שתעריף של שעה נוספת שווה ל-150% של תעריף שעה רגילה.

מבנה התאריך הוא: יום, חודש, שנה. את שעות הכניסה והיציאה נשמור במבנה מסוג time (אותו הגדרנו בתחילת הפרק). שים לב שהתוכנית אינה מלאה.

```
struct date {
            int day, month, year
            };
```

מבנה "יום עבודה"

```
struct workday {
         char  name[7];
         struct date wday;
         struct time enter, leave;
              };
```

המבנים של תאריך וזמן "הוטבעו" בהגדרת המבנה המתאר יום עבודה של עובד.

הסכימה 12.6 מתארת מבנה מסוג **struct workday**.

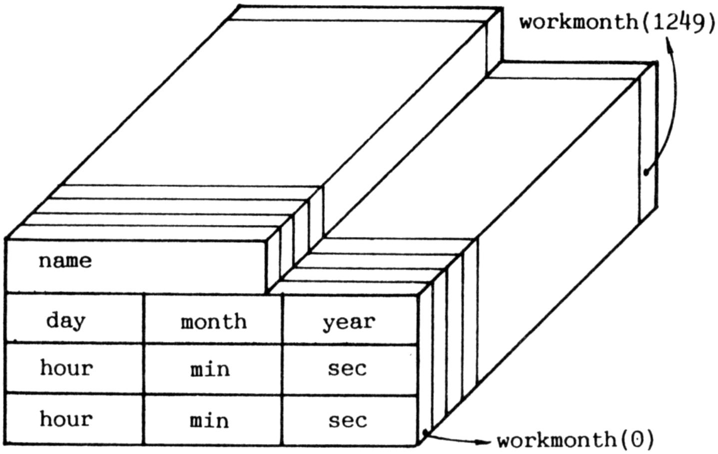
name char[7]		
day int	month int	year int
hour int	min int	sec int
hour int	min int	sec int

תרשים 12.6: *סכימה של המבנה workday*

עלינו להגדיר עבור כל עובד 25 מבנים יומיים כאלה, לכל חודש עבודה. נגדיר קבוע WORKERS, כדי לציין את מספר העובדים במפעל ונשתמש בו לציון מספר המבנים הדרוש לכל העובדים במפעל.

```
# define  WORKERS  50
struct workday workmonth [WORKERS*25];
```

הסכימה בתרשים 12.7 מתארת את המערך **workmonth**.



תרשים 12.7: *סכימה של המבנה workmonth*

תכנון ראשוני - תיאור כללי של המערכת

המערך יתמלא בנתונים במהלך החודש. בכל בוקר, יקליד כל עובד על המסוף את שמו ושעת הגעתו. מצביע למבנה workday יתקדם על פני המערך, עבור כל עובד שיגיע ויזין את הנתונים. בסיום יום העבודה יגיע כל עובד למסוף ויקליד את שמו ושעת היציאה. תוכנית המחשב תסרוק את המערך מנקודת המצביע אחורה, תאתר את המבנה עם נתוני שעת הכניסה ותעדכן בו את שעת היציאה. בסוף החודש תתבצע סריקה כוללת של כל המערך עבור כל עובד, ותוך כדי כך תחושב משכורתו.

תיאור של הפונקציות

א. `entry (w_name, today, arrive, nxt)`

הפונקציה ()entry משמשת לאחסון שעת הכניסה של העובד. הפונקציה מקבלת את הפרמטרים הבאים:

w_name שם העובד

today תאריך

arrive שעת כניסה

nxt מצביע למצביע למבנה הבא הפנוי במערך

הסיבה למצביע הכפול היא הצורך לקדם מצביע לאחר אחסון הנתונים.

ב. `wexit (w_name, left, nxt)`

הפונקציה ()wexit משמשת לעדכון שעת היציאה של העובד. הפונקציה מקבלת את הפרמטרים הבאים:

w_name שם העובד

arrive שעת היציאה

nxt מצביע למבנה הפנוי הבא

ג. `struct workday *locate (name, w)`

הפונקציה ()locate מאתרת מבנה עם שעת כניסה של העובד על פי שמו ומצביע למבנה הפנוי הבא. שיטת האיתור תהיה סריקה לאחור. הפונקציה מחזירה מצביע למבנה שאותר.

ד. `float salary (name, pay, nxt)`

הפונקציה ()salary משמשת לחישוב שכרו של עובד בסוף החודש. היא מקבלת את הערכים הבאים:

name שם העובד

pay שכרו לשעה

nxt מצביע למבנה הפנוי הבא

הפונקציה מחזירה מספר ממשי שהוא משכורת העובד.

ה. `float timedif (leave, entry)`

הפונקציה ()timedif משמשת לחישוב הפרש בין שני זמנים (כניסה ויציאה).

פונקציות השירות של התוכנית

```
/*****     chp1206.c :מתוך קובץ תקליטור     *****/

void entry(char *w_name, struct date *today,

            struct time *arrive, struct workday **nxt)
{
      strcpy((*nxt)->name, w_name);
      (*nxt)->wday.day = today->day;
      (*nxt)->wday.month = today->month;
      (*nxt)->wday.year = today->year;
      (*nxt)->enter.hour = arrive->hour;
      (*nxt)->enter.min = arrive->min;
      (*nxt)->enter.sec = arrive->sec;
      ++(*nxt);
}

void wexit(char *w_name, struct time *left,
            struct workday *nxt)
{
      struct workday *in;

      in = locate(w_name,nxt);
      in->leave.hour = left->hour;
      in->leave.min = left->min;
      in->leave.sec = left->sec;
}

struct workday *locate(char *name, struct workday *w)
{
      for (--w; (w>workmonth) && strcmp(name,w->name);--w)
        ;
      return(w);
}
```

הסבר:

א. ההקטנה של w בתחילת לולאת for נעשתה במטרה שיצביע על האיבר האחרון במערך (כאמור, w מצביע על האיבר הפנוי הבא).

ב. תנאי העצירה הוא הגעה לתחילת המערך (כשלון החיפוש), או קליטת ערך 0 מהפונקציה ()strcmp (שהוא כידוע הערך המוחזר כאשר ההשוואה מוצלחת).

ג. הפונקציה ()locate מחזירה מצביע למבנה המבוקש (במידה והחיפוש הצליח).

```
/*****      chp1206.c :מתוך קובץ תקליטור       *****/

(1)   float salary(char *name, float pay,
                       struct workday *nxt)
(2)   {
(3)     float sum, hours;

(4)     for (sum=0 ; nxt > workmonth ;) {
(5)       nxt = locate(name,nxt);
(6)       if (nxt >= workmonth) {
(7)          hours = timedif(&nxt->leave,&nxt->enter);
(8)          if (hours >8)
(9)           sum +=8*pay + (hours-8)*1.5*pay;
(10)         else
(11)          sum += hours*pay;
(12)         }
(13)      }

(14)  return(sum);
(15)  }
```

הסבר:

א. לולאת for בשורה 4 סורקת את המערך []workmonth לאחור באמצעות הפונקציה ()locate, ומאתרת את המבנים השייכים לעובד המבוקש. עצירת הלולאה נעשית כאשר המצביע למבנה הבא מגיע לתחילת המערך []workmonth.

ב. חישוב השכר מתבצע, רק אם ()locate החזירה מבנה של העובד המבוקש (שורה 6 (nxt > workmonth).

ג. timedif (שורה 7) מחזירה את סה״כ שעות עבודה באותו יום.

ד. חישוב השכר (שורות 8-11) מתבצע על ידי כפל מספר השעות בשכר לשעה, תוך תשומת לב לאפשרות של שעות נוספות, אשר התעריף שלהן גבוה ב-50% מתעריף שעה רגילה.

```
       /*****      chp1206.c :מתוך קובץ תקליטור        *****/

float timedif(struct time *leave, struct time *enter)
{
   float hours;

   hours = leave->hour - enter->hour;
   hours += (leave->min - enter->min / 60;
   hours += (leave->sec - enter->sec / 3600;
   return(hours);
}
```

כדי להפוך את כל הפונקציות שלעיל לחבילה אחת, צריך לכתוב תוכנית ראשית, דוגמת התוכנית לניהול כל-בו. התוכנית הראשית תקלוט ישירות את כל המידע מ"העולם החיצון": שמות עובדים, שעות כניסה ויציאה, שכר לשעה וקודי פעולה. על פי קוד הפעולה תקראנה הפונקציות המתאימות. אנו מציעים לקורא לתרגל את כתיבת התוכנית הראשית ופונקציות העזר.

מבנים המתייחסים לעצמם - רשימה מקושרת

הצירופים שניתן לבצע במבנים הם רבים. בפרק זה נלמד על **מבנים המתייחסים אל עצמם** (Self-referential Structure).

בסעיף הקודם ראינו, שניתן להגדיר מבנה בתוך מבנה על ידי הגדרה של משתנה מטיפוס מבנה בתוך תבנית ההגדרה של מבנה אחר. ביישומים עם מבני נתונים מורכבים, יש צורך להגדיר בתוך מבנה מסוים, מצביע למבנה מאותו טיפוס. בפרק זה תוצג לפניך דוגמה בסיסית לשימוש כזה, ובפרק הבא נראה שימושים ויישומים מורכבים יותר.

בשתי הדוגמאות האחרונות ראינו את היתרון שבהגדרת מערך של מבנים, אשר כולל קבוצת מבנים המכילים מידע בעל אופי זהה. המבנים במערכים נמצאים בזיכרון המחשב בזה אחר זה ועובדה זו אפשרה לנו לסרוק את המערך כולו, בעזרת פעולת קידום פשוטה של המצביע. בשימוש במערכים צריך להגדיר מראש את גודל המערך כדי שבזמן ההידור אפשר יהיה להקצות עבורו את המקום הנדרש לזמן הביצוע. בדוגמת חישוב השכר הגדרנו מערך עבור 50 עובדים, אך זוהי הערכה בלבד. ייתכן שהערכתנו מוגזמת ובגמר קליטת הנתונים יישארו מבנים לא מנוצלים, וייתכן שאנו זקוקים למספר רב יותר של מבנים. מבנה נתונים גמיש יותר מהמערך הוא **רשימה-מקושרת**.

רשימה מקושרת

רשימה מקושרת (Linked List) מורכבת ממבנים, אשר כוללים את שדות המידע ושדה שהוא מצביע למבנה נוסף מאותו טיפוס. המבנה הנוסף מכיל אף הוא מצביע למבנה נוסף, וכך מתקבלת שרשרת של מבנים המקושרים באמצעות מצביעים. מכיון שהמצביע הוא מטיפוס המבנה שבו הוא מוגדר, נהוג לכנות מקרה כזה **"מבנה המתייחס אל עצמו"** (או ליתר דיוק - "מבנה המתייחס למבנה מסוגו").

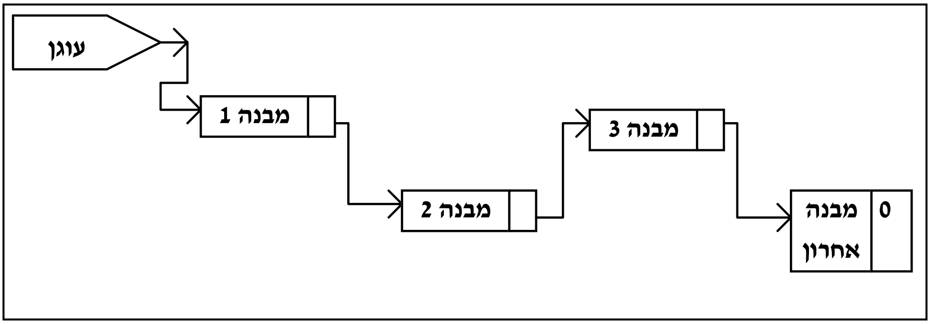
שרשרת המבנים יכולה לשמש אותנו בדיוק כמו מערך של מבנים, אלא שאין חובה להגדירם ברצף בזיכרון ואין חובה לציין את מספרם מראש. כך נוכל לצרף מבנה נוסף, על ידי הצבת כתובתו לשדה המצביע של המבנה האחרון בשרשרת.

נגדיר שני כללים:

א. המבנה האחרון ברשימה מקושרת יכיל 0 בשדה "המצביע למבנה הבא". האפס, בדומה למחרוזות, מסמן לנו את סיום השרשרת.

ב. משתנה מטיפוס "מצביע למבנה", המצביע למבנה הראשון ברשימה מקושרת, נקרא **"עוגן"**.

הסכימה בתרשים 12.8 מתארת רשימה מקושרת בת ארבעה איברים:



***תרשים 12.8:** רשימה מקושרת של מבנים*

הקצאת המשתנים נעשתה עד כה בעזרת המהדר, ועובדה זו אילצה אותנו לתכנן מראש את כל המשתנים להם נזדקק במהלך העיבוד. בשפת C קיימת פונקציה (השייכת לספריה הסטנדרטית) המבצעת הקצאה של זיכרון באופן דינמי. משמעות עובדה זו - יכולת ל**הקצות משתנים תוך כדי ריצת התוכנית**.

הפונקציה ()malloc (הקצאה, allocation) משמשת להקצאת משתנים בעת ריצת התוכנית. הערך היחיד ל-()malloc הוא מספר שלם, המכיל את כמות הזיכרון הרצויה בבתים. הפונקציה מקצה בזיכרון הפנוי במחשב את מספר הבתים כמבוקש, ומחזירה את הכתובת של תחילת קטע הזיכרון שהוקצה. כתובת זו ניתן להציב למצביע ובאמצעותו לפנות לאזור זיכרון זה, באופן דומה לגישה למשתנים שהוקצו על ידי המהדר.

בדוגמה שלהלן ניתנת השוואה בין שני סוגי ההקצאה:

א. הקצאה על ידי מהדר:

```
int  i,*p;
p=&i;
*p=10;
```

ב. הקצאה דינמית בזמן ריצה:

```
int  i,*p;
p=malloc(2);
*p=10;
```

במקרה (א) המהדר הקצה משתנה מטיפוס שלם i והתוכנית הציבה בו את המספר 10, באמצעות המצביע p. במקרה (ב) הוקצה מקום בזיכרון שגודלו כגודל הטיפוס שלם (2 בתים), ללא שם משתנה, וכתובתו אוחסנה במצביע p. כעת הגענו למצב דומה ל-(א), דהיינו - p מצביע למשתנה מטיפוס שלם, ולכן ההצבה זהה.

אלפון טלפונים - תרגול

ברצוננו לבנות תוכנית המתחזקת אלפון כתובות וטלפונים: היא תוכל לקלוט כתובת טלפון של אדם מסוים ותוכל להדפיס אלפון ממוין על פי שמות המשפחה. האלפון יהיה מורכב ממבנים, אשר כל אחד מהם מכיל שדות עבור שם פרטי, שם משפחה, טלפון וכתובת. המבנים יהיו מקושרים ביניהם באמצעות שדה מצביע נוסף. הדפסת האלפון תיעשה מהמבנה הראשון ברשימה ועד סופה. נמיין תחילה את הרשימה וכך נקבל את תדפיס האלפון כנדרש. הגדרת המבנה:

```
struct phonenum {
  char first_name[7],last_name[12],address[22],tel[11];
  struct phonenum *nxt_phone;
               } ;
```

השדה nxt_phone מוגדר כמצביע מאותו טיפוס של המבנה שבתוכו הוא מוגדר. שדה זה יצביע למבנה הבא ברשימה המקושרת. העוגן לרשימה המקושרת יוגדר כך:

```
struct phonenum *first_phone ;
```

עד כה לא הוגדרו משתנים מטיפוס phonenum, כי ביטוי זה הוא הצהרה בלבד.

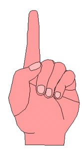
ההגדרה הראשונה היתה תבנית בלבד, והשנייה הגדירה מצביע למבנה. חסר לנו מנגנון שיאפשר להגדיר מבנה במהלך ריצת התוכנית.

כדי להקצות מקום בזיכרון בזמן ריצה עבור מבנה מטיפוס phonenum, נשתמש בפונקציית הפסידו ()sizeof. היא תיתן לפונקציה malloc את מספר הבתים הדרושים להקצאה. נגדיר כעת מבנה ראשון עבור הרשימה:

```
first_phone = malloc (sizeof (struct phonenum));
```

first_phone מצביע כעת למבנה מטיפוס phonenum (שהוקצה דינמית).

```
if (first_phone != NULL)
```



שים לב!

יש לוודא שאכן קיבלנו את שטח הזיכרון שביקשנו.

כדי להציב ערכים לשדות במבנה זה נשתמש באופרטור החץ:

```
first_phone->first_name, first_phone->last_name,
         first_phone->address,first_phone->tel;
```

במידה ונזדקק למבנה נוסף עבור אדם חדש, נוכל להקצות מבנה נוסף ולהוסיפו לרשימה על ידי אחסון כתובתו במצביע:

```
first_phone->nxt_phone ;
```

לדוגמה:

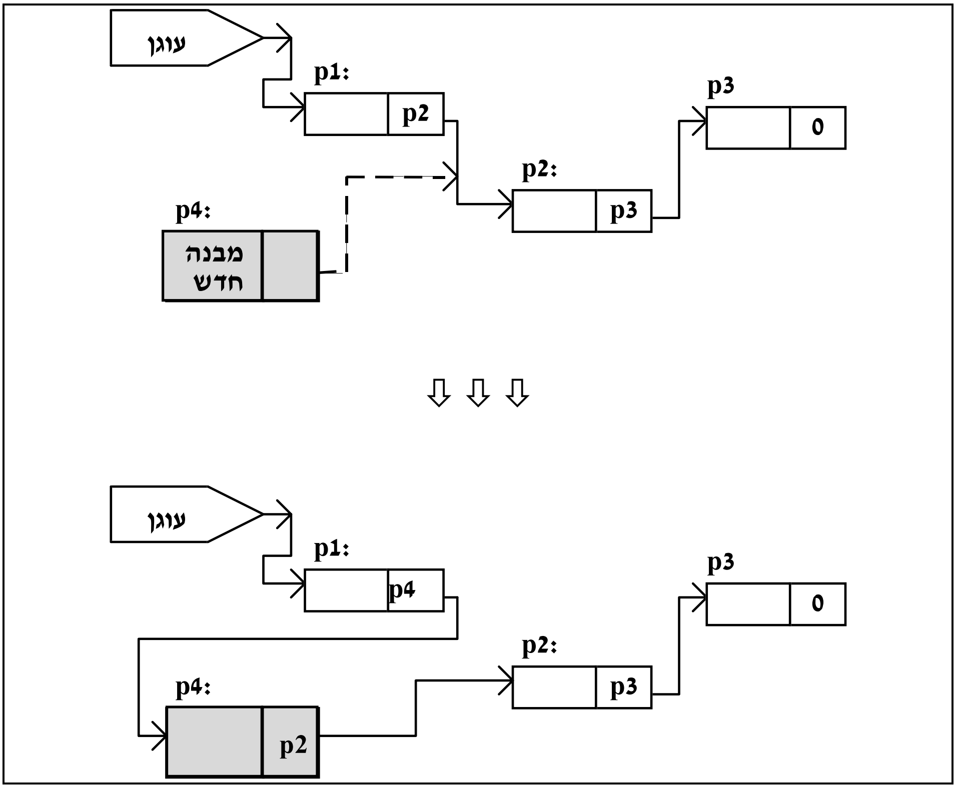
```
first_phone->nxt_phone = malloc(sizeof(struct phonenum)) ;
```

ושוב, יש לבדוק שאכן קיבלנו את שטח הזיכרון שביקשנו.

הוספת מבנה

הוספת מבנים חדשים בדרך זו איננה מבטיחה שהרשימה הסופית תהיה ממוינת על פי סדר שמות המשפחה. עלינו למצוא דרך לאתר את המקום הנכון של המבנה החדש בתוך הרשימה המקושרת, להחדירו למקום זה, ולבצע את הקישורים הנובעים מכך.

פעולה כזו מוצגת באופן סכימטי בתרשים 12.9.



***תרשים 12.9:* הוספת מבנה ברשימה מקושרת**

כפי שניתן לראות, שינינו את המצביע של המבנה p4 שאחריו יש להציב את המבנה החדש p2. בשדה הקישור של המבנה p2 רשמנו את ערך המצביע אל המבנה p3. הפונקציה שתבצע את השרשור תקבל ערך, שהוא מצביע למצביע של המבנה שאחריו יש לקשור את המבנה החדש, ותחזיר מצביע למבנה החדש. התוכנית שקוראת לה תוכל להשתמש במצביע, כדי להציב ערכים למבנה החדש.

```
      /*****      chp1207.c  :מתוך קובץ תקליטור        *****/

(1)   struct phonenum *insert(struct phonenum **after)
(2)
(3)   {
(4)     struct phonenum *save;

(5)     save = *after;
(6)     *after = malloc(sizof(struct phonenum));
(7)     if (*after != NULL)
(8)       (*after)->nxt_phone = save;
(9)     return(*after);
(10)  }
```

הסבר:

א. הערך הפורמלי after הוא מצביע למצביע שאחריו "נשתול" את המבנה החדש. כאשר מציבים ל-save את ערכו, שומרים למעשה את כתובת המבנה שמקושר אליו לפני השינוי.

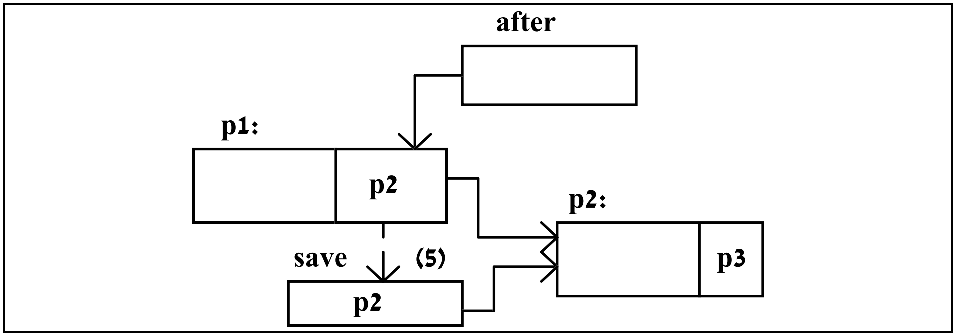
ב. הצבת תוצאות ()malloc לתוך after* משנה למעשה את הכתובת שבתוך המצביע במבנה, כך שיצביע למבנה החדש.

ג. יש לבדוק בשורה 7 אם אכן הוקצה שטח הזיכרון שביקשנו בשורה 6.

ד. לבסוף סוגרים את מעגל הקשרים על ידי הצבת הכתובת המקורית (מתוך save - שורה 8) לשדה המצביע במבנה החדש.

פעולת הפונקציה insert מורכבת למדי ועל-כן נבצע "הרצה" ידנית עבור המקרה המתואר בתרשים 12.6, ובמהלכה נסביר את המהלך הכללי של הפונקציה.

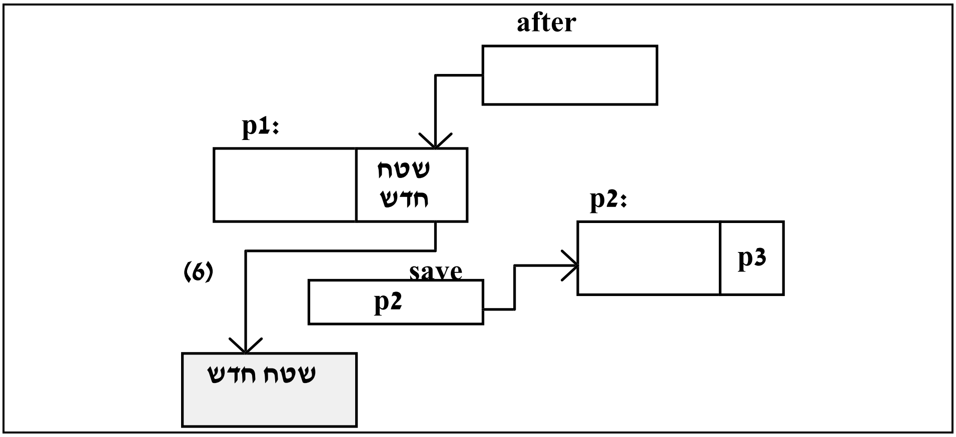
בדוגמה זו עלינו לשרשר את המבנה החדש (p4) לאחר המבנה p1. המשתנה after יצביע על שדה ה"מצביע הבא" של p1, כלומר למצביע **המצביע** ל-p2. ראה תרשים 12.10.



***תרשים 12.10:* שרשור מבנה רשימה מקושרת (1)**

בפקודה (5) הצבנו ל-save את הערך המוצבע על ידי after. כיון ש-after מצביע לשדה המכיל מצביע ל-p2, נקבל לאחר הצבה זו שגם save יצביע ל-p2.

בפקודה (6) אנו מציבים למצביע **המוצבע** על ידי after מצביע לשטח חדש המוקצה בעת הפעלת הפונקציה ()malloc. כעת, שדה ה"מצביע הבא" של p1 יצביע לשטח החדש. ראה תרשים 12.11.



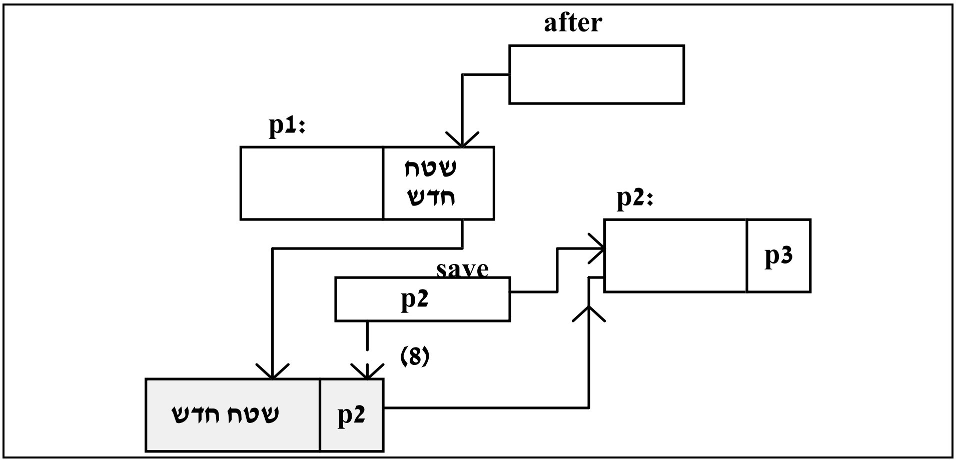
***תרשים 12.11:* שרשור מבנה רשימה מקושרת (2)**

המשתנה after מוגדר כמצביע למצביע לטיפוס struct phonenum, ולכן המהדר מכיר את השטח החדש כמבנה struct phonenum. נקבל כי

```
(*after)->nxt_phone
```

מתייחס לשדה ה"מצביע הבא" של השטח החדש.

בפקודה (8) אנו מציבים לשדה זה את תוכן המשתנה save (כלומר, מצביע ל-p2). ראה תרשים 12.12.



תרשים 12.12: *שרשור מבנה רשימה מקושרת (3)*

עתה השלמנו את תהליך השרשור, אולם נותר לנו לעדכן את שדות הנתונים בשטח החדש. אין זה מתפקידה של הפונקציה ()insert לעשות זאת, ולכן היא מחזירה מצביע למבנה החדש, כדי שפונקציה אחרת תוכל לבצע את העדכון.

בפקודה (9) מוחזר הערך **המוצבע** על ידי after. שים לב, שאכן זהו מצביע למבנה החדש. בסיום הפונקציה ()insert המשתנה הזמני save של הפונקציה ()insert נעלם, כי הוא אינו דרוש יותר.

איתור מקום ההוספה

נכתוב פונקציה שתאתר את מקום ההוספה הרצוי של מבנה חדש. נשתמש בפונקציה ()strcmp, אותה הכרנו בפרק שבו דנו במערכים. הפונקציה תחזיר מצביע למבנה **שאחריו** יש לבצע את הקישור. הפרמטר של הפונקציה הוא, כמובן, שם משפחה חדש.

```
      /*****      chp1207.c :מתוך קובץ תקליטור       *****/
struct phonenum *find_place(char *new_name)
{
  struct phonenum  *p, *save;
  char found = 0;

  for = (save=p=first_phone; p&& (!found);
                 p=p->nxt_phone)
    if (strcmp(new_name,p->last_name) <0)
       found = 1;
    else
       save = p;
  return(save);
}
```

הסבר:

א. הלולאה for מבצעת סריקה על פני הרשימה המקושרת באמצעות המצביע p. ערכו ההתחלתי של p הוא העוגן first_phone, המוגדר לגבינו כמשתנה גלובלי.

ב. המצביע save מצביע למבנה אחד לפני המבנה אותו אנו בודקים, כיון שהעצירה של הלולאה מתבצעת כאשר השם הקיים **גדול יותר** מהשם החדש.

ג. save מקבל גם הוא את ערכו של העוגן כערך התחלתי. כאן אנו מספקים תשובה למצב שהרשימה עדיין ריקה, או שהשם החדש קטן מכולם.

הוספת פרטים

הפונקציה הבאה משתמשת בשתי הפונקציות הקודמות, כדי להוסיף לרשימה פרטים של אדם חדש:

```
     /*****      chp1207.c  :מתוך קובץ תקליטור       *****/

(1)   void add_phone (char *f_nam, char *l_num,
(2)                       char *address, char *tel)
(3)   {
(4)     struct phonenum  *p;

(5)     p = find_place (l_num);
(6)     if (p==first_phone)
(7)       p = insert(&first_phone);
(8)     else
(9)       p = insert(&(p->nxt_phone));
(10)    strcpy(p->first_name, f_nam);
(11)    strcpy(p->last_name, l_nam);
(12)    strcpy(p->address, address);
(13)    strcpy(p->tel, tel);
(14)  }
```

הסבר:

א. בשורה 5 מאתרת הפונקציה את המבנה שאחריו יבוצע הקישור של המבנה החדש.

ב. במידה ו-()find_place מחזירה אפס, הרי הרשימה המקושרת עדיין ריקה, לכן מציבים מיד לאחר העוגן. שים לב שהפונקציה ()insert מקבלת כערך את **כתובת** המצביע p->nxt_phone.

ערך זה תואם את הגדרת after **כמצביע למצביע**.

הגדרת טיפוס חדש בתצורה מורכבת

שפת C מאפשרת להגדיר טיפוסים חדשים בנוסף לטיפוסים הבסיסיים שהכרנו במהלך הספר. במהלך הגדרת הטיפוס החדש משתמשים בהגדרות בסיסיות, או בהגדרות שנעשו קודם באמצעות **typedef**.

הגדרת טיפוס חדש:

```
<typedef <הגדרה> <שם טיפוס>, <שם טיפוס> ,...;
```

דוגמאות:

```
(1)  typedef int INTEGER, ANOTHER_INT;
(2)  typedef char *PTR_CHR;
(3)  typedef struct id_struct{
             PTR_CHR name;
             INTEGER id_num;
             ANOTHER_INT children[5];
             } *ID_CARD;
```

בעזרת ההגדרות שלמעלה, נוכל להכריז על משתנים:

```
INTEGER i, j, k;
PTR_CHR ptr, string;
ID_CARD id_ptr;
```

קיים דמיון רב בין הגדרת typedef להגדרות המאקרו define#. מכיון שהגדרת typedef מטופלת על ידי המהדר, בעוד שהגדרת מאקרו מטופלת על ידי קדם-מעבד, ניתן להשתמש בה להגדרות שלא ניתן לבצע באמצעות הגדרת מאקרו. לדוגמה:

```
typedef char *(*CHR_FUNC)();
```

הטיפוס CHR_FUNC הוא מצביע לפונקציה שמחזירה מצביע מטיפוס תו.

דוגמה לשימוש:

```
char      *strcmp(), *strcat();
CHR_FUNC  cmp_str_ptr, cat_str_ptr;
```

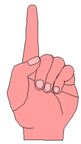
כעת נוכל להציב:

```
cmp_str_ptr = strcmp;
cat_str_ptr = strcat;
```

כפי שראינו, השימוש ב-typedef מאפשר לשפר את הקריאות וההבנה של תוכניות, מכיון שהגדרות מסובכות ומורכבות אפשר להחליף בשמות קצרים ובעלי משמעות.

רשימה מקושרת דו-כיוונית

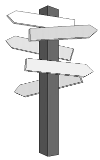
לאחר שלמדנו על עקרונות רשימה מקושרת חד-כיוונית ויישומם, נגלה שישנם מספר חסרונות, שכדי להתגבר עליהם יש צורך ברשימה מקושרת דו-כיוונית.



שים לב!

כאשר מספר הנתונים ידוע מראש, מומלץ להשתמש במערך מצביעים. כך ניתן להשיג מהירות בתהליך שליפת הנתונים, שכן ברשימה מקושרת קיימת גישה אך ורק לראש הרשומה או לסוף הרשומה. כדי להגיע לרשומה מסוימת, צריך לעבור ברצף מרשומה לרשומה, ולא על ידי גישה אקראית כמערך. יש לשים לב שגודל המערך נקבע מראש, ולכן לא נוכל להוסיף מבנים במהלך הריצה.

התוספת שיש להוסיף לרשימה דו-כיוונית הוא **מצביע למבנה הקודם** (back pointer). כך ניתן לאפשר תנועה בשרשרת המבנים קדימה ואחורה - מה שלא ניתן לבצע בשרשרת מבנים חד-כיוונית. פעולה זו מקנה מהירות בהשגת נתונים, בכך שניתן לנוע קדימה או אחורה על רשימת המבנים, ולבצע פעולות על הרשומות שקדמו לרשומה הנוכחית.



נקודת ציון!

*יתרונה הגדול של **רשימה מקושרת** (שרשרת נתונים) הוא בכך שאין צורך לדעת מראש מה מספר הרשומות שיידרשו במהלך התוכנית. כך נוכל להוסיף מבנה נתונים בכל מקום שיידרש ברשימה לאחר קבלת האישור על קבלת שטח הזיכרון שביקשנו.*

בתוכנית הבאה נלמד כיצד לטפל ברשימה מקושרת דו-כיוונית.

```
        /*****      chp1208.c  :קובץ תקליטור      *****/

רשימה מקושרת דו-כיוונית
*/
#include <alloc.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <stdlib.h>

#define MENU_LEN 7

                                    /*  (8bit=0-255) מבנה בית בזיכרון  */
typedef unsigned char byte;
```

```
/* מבנה מערך של פסיקות, ניבחר בצורה אקראית, בדיוק כמו שם וציון למשל */
typedef struct {
                  char nameintrpt[12];
                  int intrpt;
         } InterruptInfo;
/* מבנה הכולל מצביע לבלוק הנתונים (מערך פסיקות במקרה שלנו) ושני מצביעים
/* לבלוק הנתונים לפני (*prv) ואחרי (*nxt)
struct DLlist {
                  void *data;
                  struct DLlist *nxt;
                  struct DLlist *prv;
             };
/* טיפוס DLlist מסוג מבנה DLlist */
typedef struct DLlist DLlst;
/* מצביע קבוע אל תחילת הרשומה */
DLlst *DL;
/* הכרזות על הפונקציות */
void select_option(void);
char get_option(void);
void clean_memory(void);
void *DL_clean(DLlst *list);
void display_bottom_up(void);
void display_top_down(void);
void del_by_number(void);
void get_intrpt_info(void);
int intrpt_info(int *row);
DLlst *DL_del(DLlst *block, void *data);
DLlst *DL_add(DLlst *list, void *data);
void message(void);
void menu(void);
/* -----------------------------------------------------------
   M A I N - P R O G R A M
 ; ---------------------------------------------------------*/
void main(void)
 {
  menu();       // פונקציה המציירת תפריט בכתיבה ישירה למסך
  select_option();
 }
/* -----------------------------------------------------------
    F U N C T I O N S
 ; ---------------------------------------------------------*/
void select_option(void)
 {
/* נושא הפונקציה: בחירת אופציה מתפריט*/
  char option;
  while ((option = get_option()) != 'q')
```

```
      {
        switch (option)
             {
                  case 'i' : get_intrpt_info();
                             break;
                  case 'n' : del_by_number();
                             break;
                  case 'u' : display_bottom_up();
                             break;
                  case 'd' : display_top_down();
                             break;
                  case 'c' : clean_memory();
                             break;
             }
      }
  }
/* ---------------------------------------------------- */
char get_option(void)
 {
                          /*נושא הפונקציה: בדיקת תקינות בחירת האופציה מתפריט  */
  char ch, ok;

  do
   {
    ch = getch();
    ok = (ch=='n') || (ch=='u') ||
         (ch=='d') || (ch=='c') ||
         (ch=='q') || (ch=='i');
             /* ok = 1 אחרת ok = 0  מקבל את התו המתאים ch כאשר  */
    if (!ok)
      {
        gotoxy(32,24);
        printf("Error, try again\n");
      }
   }
  while(!ok);
  return(ch);
 }
/* ---------------------------------------------------- */
void clean_memory(void)
 {
                                          /*נושא הפונקציה: נקה זיכרון  */
  DL_clean(DL);
  gotoxy(1,1);
  printf("memory cleand!");
```

```
  DL=NULL;
 }
/* ------------------------------------------------------- */
void *DL_clean(DLlst *list)
 {
                                         /* נושא הפונקציה: נקה זיכרון */
  DLlst *dllNxt;
  DLlst *dllCur;

  for (dllCur = list;dllCur != NULL;)
    {
                      חייבים להעביר את המצביע לבלוק הבא אחרת נאבד*/
                                /* את הקשר, מפני שהמבנה הנוכחי נמחק
      dllNxt = dllCur->nxt;
      free(dllCur);
      dllCur = dllNxt;
    }
  return(0);
 }
/* --------------------------------------------------------
*/
void display_bottom_up(void)
 {
                           /* נושא הפונקציה: הצג רשימה מסופה ועד תחילתה */
  DLlst *list;
  DLlst *ListTail=NULL;
  InterruptInfo *IInfoCurrent;

  clrscr();
  cputs("Interrupts Information List (Bottom to
                                         up)\r\n\n");
                                              /* מגיעים לסוף הרשימה */
  for (list= DL;list != NULL;ListTail = list,list =
                                        list->nxt);
 /* חוזרים לתחילת הרשומה בעזרת המצביע prv ומקרינים על המסך את נתוני המבנה */
  for(IInfoCurrent = (InterruptInfo *)ListTail->data,
        list = ListTail;
        list != NULL && IInfoCurrent != NULL;
        list = list->prv, IInfoCurrent =
                  (InterruptInfo *)list->data)
  cprintf("Name : %12s Interrupt : %uhex\r\n",
        IInfoCurrent->nameintrpt,
        IInfoCurrent->intrpt);
  message();
 }
```

```
/* -------------------------------------------------------- */
void display_top_down(void)
 {
                                /*נושא הפונקציה: הצג רשימה מתחילתה ועד סופה  */
  DLlst *list;
  InterruptInfo *IInfoCurrent;
  clrscr();
  cputs("Interrupts Information List (Top to
                                    bottom)\r\n\n");
                              /*עוברים על הרשומה בעזרת המצביע nxt עד לסופה
                                          ומקרינים על המסך את נתוני המבנה  */
  for (list= DL, IInfoCurrent =
                 (InterruptInfo *)list->data;
           list != NULL && IInfoCurrent != NULL;
           list = list->nxt, IInfoCurrent =
                 (InterruptInfo *)list->data)
  cprintf("Name : %12s Interrupt : %uhex\r\n",
         IInfoCurrent->nameintrpt,
         IInfoCurrent->intrpt);
  message();
 }
/* -------------------------------------------------------- */
void del_by_number(void)
 {
  /* נושא הפונקציה: מחיקת מבנה ברשימה על פי המספר הסדרתי של המבנה ברשימה  */
  int intrptnum;
  InterruptInfo *IInfo;
  DLlst *tmp;           /*  מצביע למבנה אשר יצביע על תחילת הרשימה  */

  clrscr();
  gotoxy(5, 5);
  cprintf("Delete by number, enter a list block number:
");
  IInfo = NULL;
  scanf("%d", &intrptnum);

                                           /*מגיעים למספר המבנה המבוקש  */
  for (tmp = DL;intrptnum > 1;intrptnum--,
                                    tmp = tmp->nxt);
  if (tmp->data != NULL)
    {
     strcpy(IInfo->nameintrpt, (char*)tmp->data);
     DL_del(DL, IInfo);
    }
  message();
```

```
 }
/* ---------------------------------------------------- */
void get_intrpt_info(void)
 {
                                        /*נושא הפונקציה: הכנה לקליטת נתונים */
  int row = 4, rv; /*         rv - return velue */

  textbackground(CYAN);
  textcolor(BLACK);
  clrscr();
  gotoxy(5, 3);
  cputs("Interrupt name & Interrupt : (q - info end)");
  for (rv = 0;rv == 0;)
     {
      rv = intrpt_info(&row);
      row++;
      row = (row > 24) ? 4 : row;
     }
  message();
 }
/* ---------------------------------------------------- */
int intrpt_info(int *row)
 {
                                        /*נושא הפונקציה: קלט נתונים ובדיקות */
  int col=15;
  char intrptname[12];
  int intrpt, fields=0;
  InterruptInfo *IInfo;

  gotoxy(col, *row);
  clreol();                   /*   ניקוי השורה ממקום הסמן עד סופה */
                                   /*מחזיר את הערך של מספר השדות הנקלטים */
  fields = scanf("%s", intrptname);
                                              /*האם הקשנו Q או q ליציאה */
  if (stricmp(intrptname, "Q") == 0) return(1);
  fflush(stdin);                       /* המקלדת Buffer ניקוי */
  gotoxy(col + 17, *row);
  fields += scanf("%d", &intrpt);
                                   1/* = אם נקלטו יותר משתי שדות = לא תקין */
  if (fields != 2) return(1);
  IInfo = (InterruptInfo *)malloc(sizeof(InterruptInfo));
                                            /*(-1) = כאשר אין מקום בזיכרון */
  if (IInfo == NULL) return (-1);
  strcpy(IInfo->nameintrpt, intrptname);
  IInfo->intrpt = intrpt;
```

```
  if (DL == NULL)
   {                          /*  לפעם הראשונה כאשר הרשומה ריקה  */
     DL = DL_add(DL, IInfo);
     return(IInfo == NULL) ? -1 : 0;
   }
                   כאשר המבנה החדש נקלט בצורה תקינה יוחזר מצביע אליו*/
                                    ולא ל-NULL, לכן יוחזר 0, להמשך קליטה  */
  return(DL_add(DL, IInfo) == NULL) ? -1 : 0;
 }
/* -------------------------------------------------------- */
DLlst *DL_del(DLlst *block, void *data)
 {
                                          נושא הפונקציה: מחק מבנה*/  */
                   /* block - מבנה הנתונים / data - הנתונים עצמם  */
  DLlst *tmp;
  DLlst *prv;
  char found = 0;

  for (tmp = block;tmp->nxt != NULL && !found;tmp =
                                                     tmp->nxt)
    {                          /*  כאשר מצאנו את מבנה הנתונים המתאים  */
      if (strcmp((char*)data, (char*)tmp->data) == 0)
        {
         found = 1;                       /*  דואגים לצאת מהלולאה  */
                              /* מעבירים את המצביע למצביע למבנה הנתונים  */
         block = tmp;
                              מעבירים את המבנה המוצבע הבא ע"י המבנה*/
                                      המיועד למחיקה, למצביע למבנה זמני  */
         tmp = block->nxt;
                                        כך גם את המבנה המוצבע הקודם*/  */
         prv = block->prv;
                                         אם המבנה הקודם מכיל נתונים*/  */
         if (prv != NULL)

    מעבירים את מצביעו אל המבנה המוצבע הבא, על ידי המבנה המיועד למחיקה*/  */
                prv->nxt = block->nxt;
                                        כך גם את המבנה המוצבע הקודם*/  */
         if (tmp != NULL) tmp->prv = prv;
                              עתה כשניתקנו את הקשרים של המבנה בשרשרת*/
                              נשחרר מהזיכרון את נתוני המבנה המיועד למחיקה  */
         if (data && block->data != NULL)
                free(block->data);
          free(block);                   /*  ונשחרר את המבנה עצמו
*/
        }
```

```
    }
  return(tmp);
 }
/* -------------------------------------------------------- */
DLlst *DL_add(DLlst *list, void *data)
 {
                                    /* נושא הפונקציה: הוסף מבנה נתונים חדש */
  DLlst *tmp;
  DLlst *new_rec;
            /* כל עוד יש נתונים צריך להגיע לסוף הרשימה כדי להוסיף מבנה חדש */
  if (list!=NULL)
     for (tmp = list;tmp->nxt != NULL;tmp = tmp->nxt);
  else
     tmp = NULL;
                                                    /* הקצאת מקום למבנה החדש */
  new_rec = (DLlst *)malloc(sizeof(DLlst));

  if (new_rec != NULL)               /* יש מקום - מעדכנים מצביעים */
    {
     new_rec->nxt = NULL;              /* מצביע אל המבנה האחרון */
                                          /* מצביע אל מבנה הנתונים data* */
     new_rec->data = data;
                        /* מצביע אל המבנה הקודם tmp - (תחילת המבנה הקודם) */
     new_rec->prv = tmp;
                    כאשר המצביע על מבנה קודם מצביע על נתונים יש לעדכן את*/
            /* מצביע (nxt) אל תחילת המבנה ששולב, שהוא המבנה האחרון ברשימה
     if (list != NULL)
          tmp->nxt = new_rec;
    }
  return(new_rec);              /* החזרת מצביע למבנה החדש שנקלט */
 }
/* -------------------------------------------------------- */
void message(void)
 {
  gotoxy(25, 24);
  printf("Press any key to main menu...");
  getch();
  menu();
 }
/* ---------------------------------------------------------
*/
void menu(void)
 {
  int i, row = 2, col = 29;
  char *str;
```

```
  byte far *vp;         /*(VGA 0xb800).Video RAM-מצביע על ה  */
                                         /*מצביע לסוף זיכרון המסך  */
  byte far *ScrEnd = (byte far *)MK_FP(0xb800, 4000);
  char *msg[MENU_LEN] = { "     main menu       ",
                     "i - Info from user.  ",
                     "n - Delete by number.",
                     "u - Display up->down.",
                     "d - Display down->up.",
                     "c - Clean the memory.",
                     "q - Quit.            "};

  textbackground(CYAN);
  textcolor(BLACK);
  clrscr();
  for (i = 0;i < MENU_LEN; i++)
    {
        str = msg[i];
        vp = (byte far *) MK_FP(0xB800, col * 2 +
                                          row * 160);
        while ((vp < ScrEnd) && (*str != NULL))
           {
                                    /*כתוב תו ישירות לזיכרון המסך  */
                 *vp++ = *(str++);
                                  /*כתוב תכונה ישירות לזיכרון המסך  */
                 *vp++ = 127;
           }
        row += 2;
    }
 }
```

תרגילים

1. הגדר רשומה שתכיל פרטים אישיים של אדם עבור שרות שידוכים ממוחשב. נתוני הרשומה הם: שם ושם-משפחה, כתובת, מין, גיל, תחומי התעניינות. תחומי ההתעניינות מוצגים כמערך המכיל את האיברים הבאים: politics, sport, books, music, theater ו-art.

   בנה שתי רשומות person1 ו-person2 המכילות את הנתונים הבאים בהתאמה:

   - ישראל ישראלי, כינרות 20 תל-אביב, זכר, 30, מתעניין בספרות, מוסיקה וספורט.
   - נורית ברזל, בלקינד 3 תל-אביב, נקבה, 26, מתעניינת באמנות וספורט.

2. הגדר מערך של 20 רשומות כמו בשאלה 1. כתוב תוכנית שתקלוט בכל פעם נתונים של אדם חדש ותנסה להתאים לו/לה בן/בת זוג לפי הקריטריונים הבאים: מין נגדי, הפרש גילאים קטן מ-10 שנים, תחום התעניינות משותף אחד לפחות.

אם נמצא בן/בת זוג, התוכנית תדפיס את נתוני בני הזוג והקריטריונים בהם עמדו (כלומר: הפרש גילאים, תחומים משותפים).

3. נתון ביח״ר המעסיק 20 עובדים. עבור כל עובד יש לשמור את הנתונים הבאים: שם ושם משפחה, גיל, ותק, משכורת ומערך המכיל את מספר ימי החופשה שנלקחו בכל אחד מ-12 החודשים האחרונים.

כתוב תוכנית שתקלוט את הנתונים של 20 העובדים שבמפעל ותבצע את הפעולות הבאות:

א. תמיין, בעזרת מערך מצביעים, את העובדים לפי סדר האלף-בית.

ב. תמיין, בעזרת מערך מצביעים, את העובדים לפי גובה המשכורת.

ג. תמיין את העובדים לפי מספר ימי החופשה שלקח כל אחד מהם ב-12 החודשים האחרונים.

הדפס את הרשימות הממוינות.

4. כתוב את פעולות המיון שביצעת בשאלה הקודמת באמצעות פונקציה מרכזית. פונקציה זו תקבל את המצביע המתאים, המצביע אל מערך המבנים שיש למיין.

5. במערכת להנהלת חשבונות מנהלים שתי קבוצות של נתונים בחשבון הלקוח:

א. נתוני זיהוי ויתרה: שם לקוח (15 תווים), מספר חשבון, גובה האשראי שמוכנים לאפשר לו, יתרה בחשבון.

ב. תנועות של פעולות כספיות, שהתבצעו במשך החודש האחרון. תנועה מורכבת משני נתונים: מספר חשבון וסכום חיובי או שלילי (עבור הפקדה או משיכה בהתאמה).

התוכנית תקלוט עד 30 חשבונות כאלה (גמר קליטה כאשר מספר החשבון 1-). אח״כ היא תקלוט תנועות ותדפיס בכל פעם מצב חשבון. אם הלקוח חורג מסכום האשראי המותר לו, תודפס הודעת אזהרה והמשיכה לא תתבצע.

6. רשימה מקושרת של איברים הממוינים בסדר עולה. כתוב תוכנית המבצעת באמצעות פונקציות את הפעולות הבאות:

א. הכנסת איבר חדש לרשימה.

ב. הוצאת איבר מהרשימה.

ג. הדפסת הרשימה.

ד. מניית מספר האיברים ברשימה.

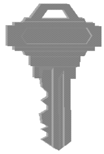
עליך להשאיר את הרשימה ממוינת לאחר כל אחת מהפעולות האלו. הגדר מבנה מתאים.

7. במסגרת הסקר הסטטיסטי הנערך כל שנה אוספים מידע מ-50 תושבי השכונה. הנתונים כוללים: מספר ת״ז, שם ושם-משפחה (20 תווים), כתובת (30 תווים), מספר ילדים, מספר חדרים בבית, גיל.

משרד הפנים מקבל מעורכי הסקר את רשימת הנבדקים, כשהיא ממוינת בסדר עולה של מספרי זהות. רשימה זו תכיל את הפרטים: שם ושם-משפחה, כתובת ומספר ת״ז.

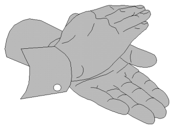
לצרכים סטטיסטיים רוצים לערוך רשימה בלי פרטים מזהים, אשר תכיל: גיל, מספר ילדים ומספר חדרים בבית. הרשימה תהיה ממוינת בסדר עולה לפי גיל, ותכיל נבדקים שגילם גבוה מ-18 שנה.

יש לכתוב תוכנית שתוציא הדפסות כנדרש בעזרת מבנה נתונים יחיד, אשר יקלוט ויבצע קישורים מתאימים בין איברי הרשימה.



רמז!

השתמש בשני שדות קישור והגדר שתי פונקציות שתבצענה קישור בהתאם לקריטריונים שהגדרת.



משימות:

תרגילים 34, 35.

פרק 13

איגודים

המבנה, אשר הכרנו בפרק הקודם, מורכב ממספר שדות בעלי טיפוס זהה או שונה, כשבכל אחד מהם ניתן להציב ערך כלשהו. לעומתו, **האיגוד** (Union) הוא משתנה בודד, אשר יכול להכיל ערכים מטיפוסים שונים, בזמנים שונים.

הגדרה של איגוד

האיגוד מאפשר להגדיר משתנה "רב משימתי", המסוגל להכיל טיפוסים שונים מבלי לדעת מה גודלו של כל אחד מהם. גישה זו תצרוך כמות מינימלית של זיכרון, עקב החפיפה במיקומם של המשתנים מהטיפוסים השונים שהוגדרו בו. המהדר הוא שדואג לבצע את הקצאת הזיכרון המתאימה.

הגדרת האיגוד זהה בצורתה להגדרת מבנה:

```
union<שם סוג האיגוד>   {
                      :
        הגדרת משתנים מטיפוסים בסיסיים
                      :
                      :
                      }   < רשימת משתני איגוד >  ;
```

תרשים 13.1: תיאור סכימטי של איגוד

הסבר:

א. מילת המפתח השמורה **union** פותחת את הגדרת האיגוד.

ב. <**שם סוג האיגוד**>: הגדרת השם הינה רשות. הגדרת שם מאפשרת להתייחס לתבנית ההגדרה של האיגוד - באופן זהה להתייחסות להגדרה של מבנה.

ג. **תבנית ההגדרה** מכילה הגדרות של משתנים בעלי טיפוסים בסיסיים או מורכבים המוכרים עד לנקודת הגדרת האיגוד. צורת ההגדרה זהה להגדרה של מבנים,

אולם באיגוד, רק משתנה אחד ויחיד מתוך תבנית ההגדרה יכול להכיל ערך ברגע נתון כלשהו.

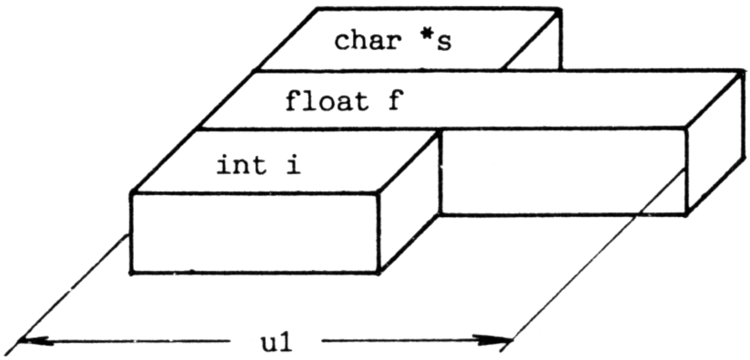
ד. <**רשימת משתני איגוד**> : רשימת משתנים הינה רשות. אם משתמשים בה, היא מגדירה משתנים מהטיפוס :

```
union <שם סוג האיגוד>
```

דוגמה:

```
union  example {
                  int  i ;
                  float f ;
                  char  *s ;
                        } u1 ;
```

בדוגמה זו הגדרנו איגוד מסוג example והקצינו משתנה-איגוד u1 מטיפוס **union example**. תבנית ההגדרה מורכבת במקרה זה משלושה שדות. באמצעותה יוכל המשתנה u1 להכיל ברגע נתון ערך כלשהו : שלם, ממשי, או מצביע למחרוזת תווים. כמות הזיכרון שיקצה המהדר עבור u1 תהיה על פי הטיפוס בעל הגודל המירבי, מכיון שהשדות באיגוד מוגדרים באותו שטח בזיכרון בחפיפה. סכימטית, נוכל לראות את u1 כמו בתרשים 13.2.



תרשים 13.2: *סכימה של מבנה איגוד (1)*

הפנייה לשדות האיגוד נעשית בדרך זהה לפנייה לשדות של מבנה (שים לב לנקודה המחברת) :

```
<שם איגוד>.<שם שדה>
```

או

```
<מצביע לאיגוד> -> <שם שדה>
```

הטיפוס של הערך המצוי באיגוד ברגע מסוים זהה לטיפוס הערך האחרון שהוצב לתוכו. פנייה לשדה מטיפוס רצוי קובעת את הטיפוס של הערך שיהיה באיגוד באותו זמן. פנייה זו, והאחריות לשימוש בערך הנכון, מוטלת על המתכנת.

לפניך סדרת פקודות להמחשה:

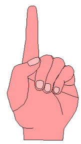
```
u1.i = 5;
printf ("u1 contains-%d\n", u1.i);
```

התוצאה:

```
u1 contains-5
```

סדרת הפקודות הבאה תניב תוצאות בלתי ניתנות לצפייה מראש:

```
u1.i = 5;
printf ("u1 contains-%f\n", u1.f);
```



שים לב!

במקרה זה, יש אי התאמה בין הערך האחרון שהוצב לאיגוד (שדה int) לבין הפנייה לאיגוד (השדה הממשי). למעשה, אין זה מדויק לומר שהתוצאה אינה ניתנת לצפייה. אם המתכנת מכיר באופן יסודי את מבנה המחשב, את גודלי הטיפוסים השונים וצורת ייצוגם בזיכרון המחשב - ניתן לנצל בכמה אופנים שונים את האיגוד בכדי לטפל בקטע הזיכרון המשותף לשדות שלו. על ידי התייחסות לשדות אחרים באיגוד (השונים מהשדה שקיבל ערך לאחרונה) ניתן לייעל פעולות שונות ולחסוך בכמות הזיכרון. הדיון בשיטות אלו חורג מנושאי הדיון שבספר זה.

שיבוץ איגוד בתוך מבנה

ניתן לשבץ הגדרת מבנה של איגוד בתוך מבנה אחר. כפי ששיבצנו מבנה בתוך מבנה, יהיה משתנה האיגוד אחד משדותיו של המבנה. צורת שימוש אופיינית תהיה יצירת מבנה המכיל איגוד, ושדה המורה לנו מה הטיפוס המצוי כרגע בתוך האיגוד. המבנה הזה יאפשר לנהל מעקב נוח אחר הטיפוס הנוכחי.

דוגמה:

```
(1)  # define INT 0
(2)  # define FLOAT 1
(3)  # define CHAR_P 2
(4)
(5)  struct var_type {
(6)       char select ;
(7)       union example u_val ;
(8)                    }  var;
```

באמצעות סדרת פקודות אלו הגדרנו משתנה מבנה בשם var, המורכב משני שדות: שדה select - שבאמצעותו נעקוב אחרי טיפוס הערך המצוי באיגוד; ושדה u_val מטיפוס union example שהוגדר קודם.

נקבע לעצמנו כלל: כל הצבה לתוך אחד משדותיו של האיגוד תלווה בהצבת ערך של סוג הטיפוס (INT, FLOAT ו-CHAR_P) לתוך שדה select. בהתאם לכך נוכל לכתוב פונקציה שתדפיס את הערך באיגוד באמצעות הפקודה printf() מתאימה:

```
(1)   p_val(struct var_type *p)
(2)
(3)     {
(4)       switch (p->select){
(5)          case INT    : printf ("%d", p->u_val.i)
(6)                        break
(7)          case FLOAT  : printf ("%f", p->u_val.f);
(8)                        break
(9)       case CHAR_P    : printf ("%s", p->u_val.s);
(10)                       break
(11)                         }
(12)    }
```

נדגים שימוש בפונקציה זו:

```
(1)   void main(void)
(2)     {
(3)       var.u_val.i = 5;
(4)       var.select = INT;
(5)       printf ("The union contains: ");
(6)       p_val(var);
(7)       printf ("\n");
(8)       var.u_val.s = "A STRING OF CHARACTERS";
(9)       var.select = CHAR_P;
(10)      printf ("THE UNION CONTAINS: ");
(11)      p_val(var);
(12)      printf ("\n");
(13)    }
```

הרצת קטע התוכנית תפיק את הפלט הבא:

```
The union contains:     5
THE UNION CONTAINS:     A STRING OF CHARACTERS
```

דוגמה לשימוש באיגוד

נבנה תוכנית המוצאת פתרונות למשוואה ריבועית, ונדגים בכך שימוש מעניין באיגוד.

למשוואה ריבועית מהצורה המנורמלת: $ax^2+bx+c=0$ קיימים שני פתרונות אפשריים:

$$x_{1,2} = \frac{-b \pm \sqrt{(b^2 - 4ac)}}{2a}$$

את הביטוי שמתחת השורש, המופיע במשוואת הפתרונות (b^2-4ac), נכנה דיסקרימיננטה ונסמן אותו ב-D. על פי ערכו של D ניתן לקבוע את אופי הפתרונות:

אם D=0 קיים פתרון יחיד: $x_1=x_2=-b/2a$

אם D>0 קיימים שני פתרונות ממשיים:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

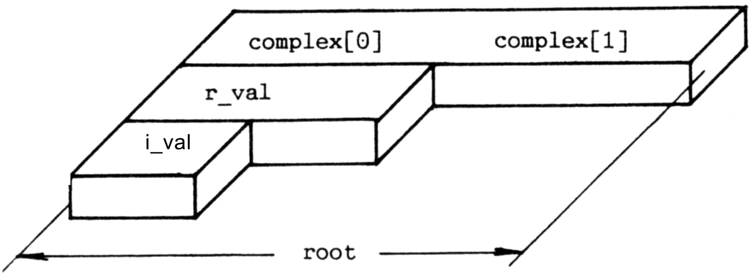
אם D<0 קיימים שני פתרונות מורכבים:

$$x_{1,2} = \frac{-b \pm \sqrt{|4ac-b^2|}\, i}{2a}$$

כדי שהתוכנית תהיה כללית ותענה על כל האפשרויות, עלינו להתייחס לכל אחת מהן. נגדיר איגוד אשר לתוכו יוכנס הפתרון בין אם הוא ממשי ובין אם הוא מורכב. נניח, שמעניין אותנו גם אם הפתרון הינו ערך שלם (עגול) או ממשי (עם שבר עשרוני). נגדיר לפיכך, את האיגוד המתאים:

```
union root {
        int  i_val;
        float  r_val;
        float  complex[2];
           }
```

תיאור סכימטי של האיגוד ניתן בתרשים 13.3.



תרשים 13.3: *סכימה של מבנה איגוד (2)*

כדי להבדיל בין סוגי הפתרונות האפשריים, "נארוז" את האיגוד בתוך מבנה המכיל שדה "סוג פתרון". הפונקציה שתחשב את הפתרון תאחסן אותו באיגוד ותעדכן את סוג הפתרון.

תכנון הפונקציות

בשלב ראשון נבנה תוכנית, שתקלוט את הערכים של המשוואה הריבועית (המקדמים a, b, ו-c) ותדפיס את פתרונות המשוואה. בשלב מאוחר יותר ניתן להרחיבה, כך שנוכל לבצע פעולות נוספות על הפתרונות בנוסף להדפסה שלהם.

פונקציות העזר הדרושות:

א. קליטת הנתונים באמצעות ()scanf.

ב. מציאת הפתרונות וסוגי הטיפוסים של הפתרונות.

ג. מציאת אופי הפתרונות על פי הדסקרימיננטה (D).

ד. הדפסת התוצאות.

התוכנית

```
      /*****      chp1301.c :קובץ תקליטור      *****/
(1)    #include <stdio.h>
(2)    #include <math.h>

(3)    #define REAL    0
(4)    #define COMPLX  1
(5)    #define ROUND   2

(6)    union root {
(7)      int i_val ;
(8)      float r_val ;
(9)      float complex[2] ;
(10)     } ;

(11)   struct roots {
(12)     char res_type ;
(13)     union root res_val ;
(14)     } res [2] ;

(15)   char typeof(), typerl() ;

(16)   main(void)
(17)   {
(18)     float a, b, c ;

(19)     printf("Enter a, b, c coefficients for quadratic
                                  equation ax2+bx+c=0\n");

(20)     scanf("%f %f %f", &a, &b, &c) ;
(21)     compute (a, b, c) ;
```

```
(22)      print (a, b, c) ;
(23)    }
(24)    compute (a, b, c) ;
(25)      float a, b, c ;
(26)    {
(27)      float D, rl, imag ;
(28)      char t;

(29)      D = b*b - 4*a*c ;

(30)      switch (t=typeof(D))
(31)      {
(32)        case COMPLX:
(33)           res[0].res_type = t ;
(34)           rl = -b / (2*a) ;
(35)                     /* D שלילי, ולכן נחשב שורש של מספר חיובי */
(36)           imag = sqrt(-D) / (2*a) ;
(37)           res[0].res_val.complex[0] = rl ;
(38)           res[0].res_val.complex[1] = imag ;
(39)           break;
(40)        case REAL:
(41)           rl = -b + sqrt(D) / (2*a) ;
(42)           res[0].res_type = (t=typerl(rl)) ;
(43)           switch (t) ;
(44)            {
(45)            case REAL:
(46)              res[0].res_val.r_val = rl ;
(47)              break;
(48)            case ROUND:
(49)              res[0].res_val.i_val = (int) rl ;
(50)              break;
(51)            }
(52)           rl = (-b - sqrt(D)) / (2*a) ;
(53)           res[1].res_type = (t=typerl(rl)) ;
(54)           switch (t) ;
(55)            {
(56)            case REAL:
(57)              res[1].res_val.r_val = rl ;
(58)              break;
(59)            case ROUND:
(60)              res[1].res_val.i_val = (int) rl ;
(61)              break;
(62)            }
(63)           break;
(64)      }
(65)    }
(66)    char typeof (d)
(67)      float d ;
```

```
(68)  {
(69)     if (d<0)
(70)        return(COMPLX);
(71)     return(REAL);
(72)  }

(73)  char typerl (r)
(74)     float r ;
(75)  {
(76)     int i ;
(77)     i = (int) r ;
(78)     if (i == r)
(79)        return(ROUND);

(80)     return(REAL);
(81)  }
(82)  print (a, b, c)
(83)     float a, b, c ;
(84)  {
(85)     int i ;
(86)     printf("(%f)x2 + (%f)x + (%f) = 0\n",a, b, c);
(87)     printf("X1,2 = ") ;

(88)     switch (res[0].res_type)
(89)        {
(90)        case COMPLX:
(91)            printf("(%f + i%f), (%f - i%f)\n");
                         res[0].res_val.complex[0],
                         res[0].res_val.complex[1],
                         res[0].res_val.complex[0],
                         res[0].res_val.complex[1]);
(92)            break ;
(93)        case REAL: case ROUND:
(94)            for (i=0 ; i<1 ; ++i)
(95)             {
(96)             switch (res[i].res_type)
(97)               {
(98)               case REAL :
(99)                  printf("%f", res[i].res_val.r_val);
(100)                     break;
(101)              case ROUND:
(102)                 printf("%d",res[i].res_val.i_val);
(103)                     break ;
(104)            }
(105)       printf((i==0) ? ", " : "\n") ;
(106)       }
(107)    }
(108) }
```

הסבר:

א. בשורות 5-3 השתמשנו במנגנון "ההגדרה" #define עבור שלושת הטיפוסים של הפתרונות.

ב. מכיון שייתכנו שני פתרונות, הגדרנו משתנה מבנה בשם res כמערך של שני מבנים res[0] ו-res[1]. כל מבנה כזה מכיל איגוד עבור הטיפוסים האפשריים של הפתרון ושדה סוג פתרון בשם res_type.

ג. בשורה 15 מבוצעת הגדרה מקדימה של שתי פונקציות עזר המחזירות ערך מטיפוס תו. הסיבה לכך נובעת משימוש בפונקציות לפני ההגדרה שלהן.

ד. התוכנית קולטת מהמסוף שלושה מקדמים של משוואה ריבועית, מחשבת את הפתרונות באמצעות הפונקציה compute() ומדפיסה את התוצאות באמצעות הפונקציה print().

ה. הפונקציה compute() מחשבת את הדסקרימיננטה (D). הפרמטרים שלה הם ערכי המשוואה. על פי סוג הדסקרימיננטה מתפצל המשך החישוב באמצעות הפקודה switch. שים לב ש-switch (שורה 30) פועל על תוצאות הפונקציה typeof(D), ולא ישירות על D.

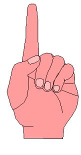
ו. במידה והפתרונות מורכבים, מחושבים ערכים רק עבור res[0]. הפתרון השני res[1], יהיה זהה בערכיו לפתרון הראשון, בחילוף סימן האיבר הדמיוני.

ז. עבור פתרון לא מורכב מחושבים שני פתרונות. באמצעות הפקודה switch הפועלת על תוצאת הפונקציה typerl(rl) (שורות 42, 53), מאחסנים את התוצאה בשדה השלם, או בשדה העשרוני של האיגוד.

ח. הפונקציה print() בשורה 82, מקבלת את ערכי המשוואה. הפונקציה בודקת את תוכן שדה סוג התוצאה במערך מבנה התוצאות (שורה 86) ועל פיו מדפיסה את התוצאות מתוך השדות המתאימים של האיגוד.

ט. עבור תוצאה מורכבת יתקבלו התוצאות רק מהמבנה הראשון ובחילוף סימן ביניהם (שורה 91).

י. כאשר התוצאה אינה מורכבת יודפסו שני הפתרונות באמצעות לולאה.



שים לב!

בשורה 105 נמצאת פקודה זו:

```
printf ((i==0) ? ", " : "\n");
```

*על פי ערכו של i (מספר הפתרון) היא בוחרת בסימן **שיסיים** את שורת הפתרון.*

דוגמאות הרצה:

```
Enter a, b, c coeficients   for
                 quadratic  equation ax^2+bx+c=0

1 -8.5 18
(1.000000)x^2 + (-8.500000)x + (18.000000) = 0
X1,2 = 4.500000, 4
```

```
Enter a, b, c coeficients   for
                 quadratic  equation ax^2+bx+c=0

2 -2 5
(2.000000)x^2 + (-2.000000)x + (5.000000) = 0
X1,2 = (0.500000 + i1.500000), (0.500000 - i1.500000)
```

```
Enter a, b, c coeficients   for
                 quadratic  equation ax^2+bx+c=0

1.2 -3.45 -0.55
(1.200000)x^2 + (-3.450000)x + (-0.550000) = 0
X1,2 = 3.026443, -0.151443
```

תרגילים

1. הגדר מבנה איגוד בשם employee המכיל שני מבנים ומפתח מסוג תו.

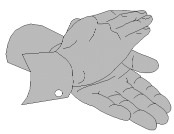
- המבנה הראשון מיועד לצוות הניהולי של המפעל ומכיל את הפרטים הבאים: שם ושם-משפחה, גיל, מספר שנות ותק, השכלה (מספר שנות לימוד), אחוז מניות במפעל, דרגה ניהולית (מספר מ-1 עד 20).
- המבנה השני מיועד לפועלי הייצור ומכיל את הפרטים: שם ושם משפחה, גיל, מספר שנות ותק, סוג התמחות מקצועית (1-100) ושיוך מחלקתי (1-10).

2. הגדר מבנה employee שיענה על הדרישות שבשאלה 1.

מהם ההבדלים בין שתי הדרכים להגדרת המבנה הזה.

3. כתוב תוכנית שתקלוט רשימה של 20 עובדים ותכניס אותם למערך של איגודים מסוג employee.

בגמר הקלט, התוכנית תמיין את הרשימה לפי מפתח הצוות. אחר כך היא תדפיס שתי רשימות ממוינות לפי אלף-בית עבור צוות המנהלים ועבור עובדי הייצור.



משימות:

תרגילים 36-38.

פרק 14

נושאים מתקדמים

תוכניות עם פרמטרים

הסתכל בתוכנית לפתרון משוואה ריבועית, בפרק העוסק באיגודים. התוכנית הראשית ()main מורכבת רובה ממשפטים שתפקידם לקלוט את נתוני המשוואה. שפת C מספקת מנגנון, שמאפשר לתוכנית הראשית לקבל נתונים במסגרת **שורת הפקודה** (Command-line Arguments).

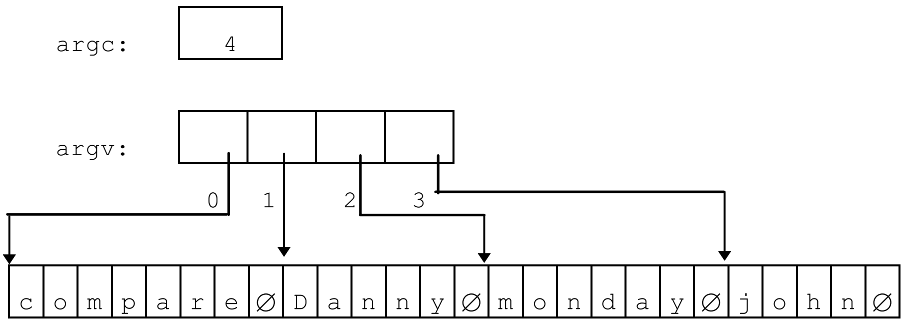
לתוכנית ()main יש שני פרמטרים פורמליים : **argc** ו-**argv**.

```
void main(int argc, char *argv[])
{
...
}
```

הערך **argc**, מכיל את מספר הנתונים שנכללו בשורת הפקודה, כולל שם התוכנית. הערך **argv**, הוא מערך של מצביעים למחרוזות הנתונים. אם התוכנית נמצאת בקובץ compare.c, ונפעיל אותה כך :

```
compare Danny monday john
```

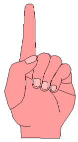
יתקבל מבנה, אשר נוכל להציג בצורה סכימטית, שבה שם התוכנית הוא הערך הראשון המוצבע על ידי מערך המצביעים argv.



***תרשים 14.1:** סכימת הפרמטרים הפורמליים של main*

בהפעלת תוכנית ללא ערכים נקבל, אם כך :

```
argc = 1
argv[0] -> <שם התוכנית>
```



שים לב!

*לפני שנראה את השימוש בנתוני שורת הפקודה, נדגיש ש**כל** הנתונים יתקבלו כמחרוזות (argv - מערך מצביעים למחרוזות). אם התוכנית זקוקה לערכים מספריים, יש לבצע המרה ממחרוזת לטיפוס הרצוי.*

בספריה הסטנדרטית של שפת C נמצאות פונקציות שמבצעות המרות בסיסיות :

- המרה ממחרוזת לערך שלם — **atoi** - ascii to integer
- המרה ממחרוזת לערך ממשי — **atof** - ascii to float

הפונקציות האלו מקבלות מצביע למחרוזת ומחזירות מספר מהטיפוס המתאים.

דוגמה:

התוכנית לפתרון המשוואה הריבועית $ax^2+bx+c=0$ זקוקה לשלושה ערכים מטיפוס ממשי המגדירים את הקבועים a, b ו-c. לפניך שילוב מנגנון הפרמטרים עם התוכנית הזו.

```
        /*****      chp1401.c  :קובץ תקליטור      *****/

(1)     #include <stdio.h>
(2)     #include <stdlib.h>

        /* --------------------------------------------------
         ;  M A I N - P R O G R A M
         ; ---------------------------------------------- */
(3)     void main(int argc, char *argv[])
        {
(4)       float a, b, c;

            /* לא התקבלו מספיק פרמטרים. כזכור, עלינו לקבל בשורת הפקודה את
             משתני המשוואה הריבועית וגם את שם התוכנית, ובסה"כ 4 ארגומנטים.  */
(5)       if (argc != 4)
           {

(6)         printf("Non valid input - try again...\n");
(7)         exit(0);                       /* סיים את התוכנית כאן  */
           }
(8)       a = atof(argv[1]);
(9)       b = atof(argv[2]);
(10)      c = atof(argv[3]);
```

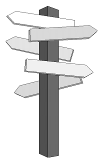
```
(11)      printf("Received : %f, %f, %f\n", a, b, c);
        }
```

הסבר:

א. בשורה 4 הגדרנו את משתני המשוואה הריבועית. כפי שציינו, הפונקציה atof() מחזירה ערך מטיפוס ממשי.

ב. בשורה 5 בודקים אם התוכנית אכן הופעלה עם מספר הנתונים הדרוש. בתוכנית זו אין אנו מבצעים בדיקות לתקינות והתאמה של הערכים.

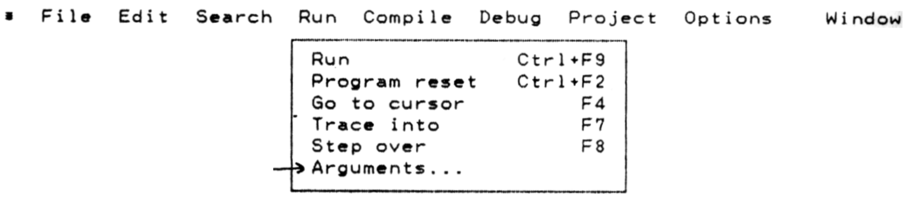
ג. בשורות 8-10 מציבים למשתני המשוואה את התוצאות המתקבלות מהמרת המחרוזות המוצבעות על ידי מערך argv. שים לב, שהנתון הראשון מתקבל מהמחרוזת המוצבעת על ידי argv[1], שהוא האיבר השני במערך. האיבר הראשון argv[0] מצביע על שם התוכנית.



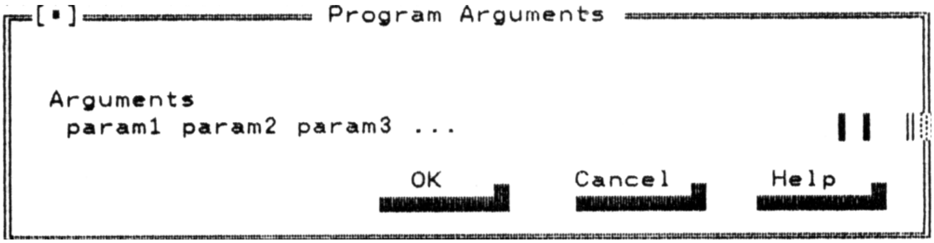
נקודת ציון!

בעת הבדיקות של התוכנית אין צורך לצאת למערכת DOS כדי להעביר ארגומנטים לתוכנית. ניתן לעשות זאת דרך מערכת הפיתוח.

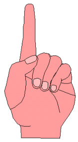
תרשימים 14.2 ו-14.3 מתארים כיצד להעביר ארגומנטים לתוכנית דרך מערכת פיתוח.



תרשים 14.2: *שלב ראשון בהעברת ארגומנטים*



תרשים 14.3: *שלב שני בהעברת ארגומנטים*



שים לב!

יש אפשרות לפרמטר נוסף במסגרת הגדרת main

```
void main (int argc, char *argv[],
                    char *envr[])
```

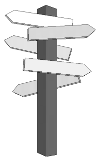
פרמטר זה מייצג את משתני הסביבה כגון : Prompt, PATH וכו'.

מצביעים לכתובת המסך

למסך יש כתובת מיוחדת בזיכרון, אשר ניתן לקרוא ממנה ולכתוב אליה בגישה ישירה על ידי מצביעים.

כתובת מסך מונוכרום : Oxb0000

כתובת מסך צבעוני : Oxb8000



נקודת ציון!

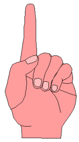
הקידומת Ox מציינת ספרה הקסדצימלית.

אופן הגישה למקום זה נעשית על ידי הכוונת מצביע לתחילת הכתובת או למקום כלשהו במסך, טיול על פני כתובת זו וביצוע קריאה/כתיבה במהלך הטיול.

כתובת המסך אינה נמצאת באזור הזיכרון של התוכנית, ולכן, נדרש **מצביע רחוק** (far pointer).

השיטה לקבלת מצביע לכתובת המסך:

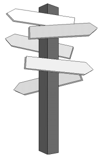
```
char far *ch_ptr_scr = MK_FP (0xb800, 0);
```



שים לב!

MK_FP יוצר מצביע רחוק והוא מוגדר ב-<dos.h>.

```
mk_fp -> make far pointer
```



נקודת ציון!

המצביע ch_ptr_scr מצביע אל התו הראשון בכתובת המסך המוקרן בשורה הראשונה, עמודה ראשונה.

כיצד מחולק המסך

המסך מחולק ליחידות של תו ותכונה

תכונה	תו

כאשר התכונה מייצגת את צבעו של התו ואת תכונתו. זאת אומרת התו הוא צהוב ומהבהב, למשל.

מכיון שבמסך יש 80 עמודות ו-25 שורות, ניתן להקרין בו 2000 תווים. מכיון שכל תו מורכב משני תאים (התו עצמו ותכונתו) נגלה בחישוב פשוט שזיכרון המסך מורכב למעשה מ-4000 תאים. 2000 תאים עבור אחסון התווים ו-2000 עבור אחסון תכונות התווים.

התוכנית הבאה ממחישה כתיבה ישירה לזיכרון המסך. אנו נכתוב על המסך מחרוזת, במיקום y,x ובצבע הנדרש.

```
        /*****      chp1402.c :קובץ תקליטור      *****/

                                 /*  כתיבה ישירה של מחרוזת לזיכרון המסך  */

#include <stdio.h>
#include <conio.h>
#include <dos.h>

    /*  (VGA) יצירת קבועים המצביעים על תחילתה וסופה של כתובת מסך צבעוני  */
#define SCREEN_START MK_FP(0xb800,0)
#define SCREEN_END   MK_FP(0xb800,4000)

void put_msg(char *str, int row, int col, int attr);
/* ----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ------------------------------------------------------*/
```

```
int main(void)
 {
   clrscr();
   put_msg("avi boots", 12, 35, 0x30);
   printf("\nPress any key\n");
   getch();
   return(0);
 }
/* -----------------------------------------------------
 ;  F U N C T I O N S
 ; ---------------------------------------------------*/
void put_msg(char *str, int row, int col, int attr)
 {
  char far *str_pos = (char far*)SCREEN_START;

            /*  אנו מורידים 1 ממשתני row, col מפני שטבלת הזיכרון מתחילה ב-0
            אורכה של שורה אחת במסך הוא 80 תווים. יחידת תו מורכבת מ- 2 בתים: תו
            ותכונה, לכן אורכה הוא 160. עלינו להכפיל את מספר השורה באורכה כדי להגיע
            לשורה המתאימה. לאחר שהגענו לשורה המתאימה, עלינו להגיע לעמודה
            המתאימה. עמודה מכילה 25 תווים ושוב עלינו להכפילה ב-2 (תו, תכונה).
            בסופו של דבר, המצביע str_pos יצביע בדיוק בנקודה הרצויה (col,).  */
  str_pos += (row - 1) * 160 + (col - 1) * 2;
  while (str_pos < SCREEN_END && *str)
      {
                         /*    כעת מציבים ביחידת תו את התו עצמו ואת תכונתו
                          עד לסוף המחרוזת, או עד לסוף המסך.  */
         *(str_pos++) = *(str++);        /*  הצבת התו המתאים  */
         *(str_pos++) = attr;          /*  הצבת התכונה המתאימה*/
      }
 }
```

חיפוש ומיון בעזרת פונקציות ספריה

מערכת **ספריות הפיתוח** מספקת פונקציות מיון וחיפוש חזקות במיוחד, אשר תוכננו ונכתבו עבורנו בחברת Borland. כל מה שנדרש מאיתנו הוא ללמוד היטב כיצד פועלות פונקציות אלו ולשלבן בתוכניתנו, כאשר דרוש.

מיון מהיר

הפונקציה qsort מבוססת על **אלגוריתם החיפוש המהיר** (quicksort algorithm). לצורך הפעלתה דרושים הפרמטרים הבאים:

```
void qsort(void *base, size_t nelem, size_t width,
         int (*tcmp) (const void*?, const Void *)?);
```

***base** מצביע מערך למיון.

nelem מספר הכניסות בטבלה.

width מספר הבתים בכל כניסה.

***fcmp** מצביע לפונקציית השוואה אשר אותה יגדיר המשתמש.

בתוכנית הבאה נדגים שימוש בפונקציה qsort.

```
        /*****     chp1403.c :קובץ תקליטור       *****/
                                    /* (qsort מיון מהיר (שימוש בפונקציה */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

#define LEN 5

char list[LEN][4] = { "cat", "car", "cab", "cap", "can" };

int sort_function(const void *a, const void *b);
void print_array(char *msg);

/* ----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; --------------------------------------------------------*/
int main(void)
 {
  clrscr();
  print_array("\nbefor \"qsort\" : \n");
  qsort((void *)list, LEN, sizeof(list[0]),
                                        sort_function);
  print_array("\nafter \"qsort\" : \n");
  printf("\nPress any key\n");
  getch();
  return (0);
 }
/* ----------------------------------------------------------
 ;  F U N C T I O N S
 ; --------------------------------------------------------*/
int sort_function( const void *a, const void *b)
 {
  return (strcmp((char *)a, (char *)b) );
 }
void print_array(char *msg)
```

```
{
  int i;

  printf("%s\n", msg);
  for (i = 0; i < LEN; i++)
        printf("%s\n", list[i]);
}
```

תוצאות ריצת התוכנית:

```
befor "qsort" :

cat
car
cab
cap
can

after "qsort" :

cab
can
cap
car
cat

Press any key
```

חיפוש בינארי

על עקרונות החיפוש הבינארי למדנו בפרק על מערכים. עקרונות אלה תורגמו לשפת C ונכתבו עבורנו על ידי אנשי Borland. כל שנותר לנו הוא ללמוד כיצד להשתמש בפונקציה bsearch העושה שימוש בעקרונות אלה. לצורך הפעלתה נדרשים הפרמטרים הבאים:

```
void *bsearch      (const void *key, const void *base,
                   size_t nelem, size_t width,
                   int (8fcmp) (const void*, const void*));
```

key* מצביע לערך שמחפשים.

שאר הפרמטרים הוצגו בהסבר לפונקציה qsort, והם נכונים גם לפונקציה זו.

בתוכנית הבאה נדגים שימוש בפונקציה bsearch.

```
        /*****      chp1404.c  :קובץ תקליטור       *****/

                              /*  (bsearch חיפוש מהיר (שימוש בפונקציה  */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

                                  /*  תהליך חישוב מספר האלמנטים במערך  */
#define NELEMS(arr) (sizeof(arr) / sizeof(arr[0]))
#define LEN 6

typedef int (*fptr)(const void*, const void*);
int numarray[LEN] = {123, 145, 512, 627, 800, 933};

int lookup(int key);
int numeric(const int *p1, const int *p2);
void print_array(char *msg);

/* ---------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; -------------------------------------------------------*/
int main(void)
 {
  clrscr();
  print_array("view the array : \n");
  if (lookup(512))
        printf("\n512 is in the table.\n");
  else
        printf("\n512 isn't in the table.\n");
  printf("\nPress any key\n");
  getch();
  return(0);
 }
/* ---------------------------------------------------------
 ;  F U N C T I O N S
 ; -------------------------------------------------------*/
int lookup(int key)
 {
  int  *itemptr;

  itemptr = (int *) bsearch(&key, numarray,
          NELEMS(numarray), sizeof(int), (fptr)numeric);
  return(itemptr != NULL);
 }
```

```
int numeric(const int *p1, const int *p2)
 {
   return(*p1 - *p2);
 }

void print_array(char *msg)
 {
   int i;

   printf("%s\n", msg);
   for (i = 0; i < LEN; i++)
         printf("%d\n", numarray[i]);
 }
```

תוצאות ריצת התוכנית:

```
view the array :

123
145
512
627
800
933

512 is in the table.

Press any key
```

פונקציה בעלת מספר משתנה של פרמטרים

כבר השתמשנו בפונקציה printf או scanf ולמדנו שתבנית הפרמטרים של הפונקציות הללו משתנה בסוגי המשתנים (char ,float ,int) ובמספרם. בשפת C יש לנו אפשרות לכתוב פונקציה משלנו, בעלת מספר משתנה של פרמטרים. במקום להציגם למסך כב-printf, אנו פשוט נאסוף אותם אל הפונקציה ולעבד אותה לפי צרכינו.

העברת תבנית הפרמטרים אל הפונקציה נעשית בעזרת מחרוזת. את המחרוזת הזו יש להמיר למאגר. תהליך ההמרה מתבצע בעזרת המאקרו va_start אל תוך מערך va_list המחזיק את המידע הנדרש אודות הארגומנטים. המאקרו והמערך האלו מוגדרים במהדר Borland. מערך זה עובר תהליך עיבוד או הדפסה לפי צרכינו.

שים לב!

תנאי חשוב בצורת הכתיבה הפרמטרית הוא שלוש הנקודות בסוף הפונקציה הפרמטרית. לדוגמה:

```
my_printf (char *str, ...);
```

בתוכנית הדוגמה הבאה נלמד כיצד לכתוב פונקציה בעלת מספר משתנה של פרמטרים.

```
        /*****      chp1405.c :קובץ תקליטור       *****/

                          /*  (ASCII הדגמת כתיבה פרמטרית (הקרנת טבלת  */

#include <stdio.h>
#include <conio.h>
#include <stdarg.h>
#include <dos.h>

typedef unsigned char byte;

void my_printf(byte col, byte row, byte attr, char *str,
                                                      ...);

/* ---------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; -----------------------------------------------------*/
int main(void)
 {
  byte attr = 0x30; /* Color Black(0x00) on Cyan(0x30) */
  int i = 0;
  textcolor(BLACK);
  textbackground(LIGHTGRAY);
  clrscr();
  gotoxy(30, 1);
  printf("A S C I I   T A B L E");
  for (i = 0; i < 256; i++)
        my_printf((i % 13) * 6 + 1, (i / 13) + 1, attr,
                             "%3d:%c", i, i);
  gotoxy(22, 23);
  printf("printf(...) - הדגמת כתיבה פרמטרית");
  gotoxy(25, 25);
  printf("Written in 1996");
  getch();
  return(0);
 }
```

```
/* -----------------------------------------------------------
 ;  F U N C T I O N S
 ; -------------------------------------------------------------*/
void my_printf(byte col, byte row, byte attr,
                                      char *str, ...)
 {
      /*,va_arg - (מאקרו) מערך המחזיק מידע הנדרש ע"י שהוא גם מאקרו המבצע
                    */ את רשימת המשתנים. זאת אומרת ממיר את הפרמטרים שב- (*str).
  va_list parameter;
                      /* .(*str) המחזיק את תבנית המחרוזת Buffer */
  char outbuf[255];
                                          /* .Buffer -מצביע על ה */
  char *outp;
                                       /* .Video RAM -מצביע על ה */
  byte far *vp;
      /*.(*str)-ממיר את הפרמטרים (מאקרו הקובע מצביע על המשתנה הראשון שב */
  va_start(parameter, str);
                  /* buffer-אל ה (*str) - שולחת את תבנית הפרמטרים אשר ב */
  vsprintf(outbuf, str, parameter);
  vp = (byte far *) MK_FP(0xB800, col * 2 + row * 160);

  for (outp = outbuf; *outp ;)
     {
        *vp++ = *(outp++);          /* video RAM -כתוב תו ל */
        *vp++ = attr;             /* video RAM-כתוב תכונה ל */
     }
 }
```

תוצאות ריצת התוכנית:

לא ניתן להציג כאן את תוצאות ריצת התוכנית. התוצאה המתקבלת היא טבלת ASCII, הכוללת תווים מיוחדים. הרץ אותה בעצמך, בחן את הפלט ובדוק לעומת הפעולות המתבצעות בגוף התוכנית.

הקצאת זיכרון דינמית

הקצאת זיכרון דינמית (memory allocation) נועדה למניעת בזבוז של משאבי זיכרון. העיקרון המנחה של הקצאה דינמית הוא הקצאת שטח זיכרון שגודלו ניתן לשינוי במהלך התוכנית לפי הצורך. נחזור לנושא מערכים, ונגדיר מערך בן 100 מחרוזות:

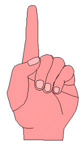
```
char str[100] [80];
```

המערכת הקצתה עבורנו מקום ל-100*80 תווים. כלומר, 800 תווים בזמן הידור. מה יקרה אם אורך המחרוזות במקרים מסוימים קטן מ-80? בזבזנו מספר תווים לשווא; ואם המערך אינו מלא עד סופו? בזבזנו הרבה יותר תווים לשווא. כאשר נדרש להוסיף מחרוזות במהלך העבודה (כשהתוכנית רצה) אנו בבעיה.

פתרון הבעיה הוא הקצאה דינמית של זיכרון. כלומר, הקצאת זיכרון מדויקת, בדיוק לפי מספר התווים הנדרשים, או לפי הצורך במהלך ריצת התוכנית. כך ניתן למנוע חלק ניכר מבזבוז זיכרון הנגרם על ידי שימוש במערך נתונים רגיל.

שימושים שכיחים

- ❖ הגדרת טבלאות "דינמיות" אשר שטח הזיכרון שהן תופסות ידוע מראש ומשתנה על פי הצרכים במהלך ריצת התוכנית.
- ❖ מבנים של נתונים בזיכרון (עליהם נדבר בפרק העוסק במבנים).
- ❖ כל תהליך או פעולה הדורשת שטח זיכרון, אשר גודלו אינו ידוע בזמן ההידור.



שים לב!

אישור קבלת שטח הזיכרון מותנה במערכת ההפעלה. ***תמיד*** *יש לוודא שניתן אישור ממערכת ההפעלה, כי אם לא נבקש, אנו עלולים לדרוס שטח זיכרון המוקצה לצרכים אחרים, והתוצאות בעל מקרה הינן הרות אסון.*

הספריה אשר מטפלת בהקצאת זיכרון היא <alloc.h> ובה מרוכזות הפונקציות המתאימות.

עתה נערוך היכרות עם הפונקציות המיועדות להקצאת זיכרון.

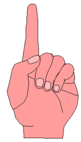
sizeof()

הפונקציה sizeof() היא אופרטור ומילה שמורה, המחזירה גודל של משתנה או טיפוס משתנה בבתים.

```
sizeof (<טיפוס>)
sizeof (משתנה)
```

לדוגמה

```
int i;
char ch;
char s_user [20];
int  i_number [10];
int  result;
result = sizeof(i);        --> result = 2  (int  = 2 bytes)
result = sizeof(ch);       --> result = 1  (char = 1 bytes)
result = sizeof(s_user);    --> result = 20
                                          (char = 1*20 bytes)
result = sizeof(i_number); --> result = 20
                                          (int  = 2*10 bytes)
```



שים לב!

*מומלץ מאוד לבדוק גודל של משתנים בעזרת sizeof(), מפני שגודלם של טיפוסי int תלויים במערכות השונות, ולעיתים שונים ממערכת למערכת. לכן, כדי לשמור על **יבילות** (portability) בין מערכות שונות. יש לבצע בדיקה זו ולתאם את המהלכים הבאים הקשורים לטיפוס בהתאם לתוצאת scanf.*

הפונקציה ()malloc

הפונקציה ()malloc - הקצאת בלוק זיכרון.

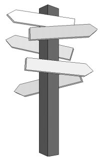
מבנה הפונקציה:

```
void *malloc (size_t size);
```

size_t קבוע של בורלנד.

size מספר בתים להקצאה.

הפונקציה מאתרת שטח זיכרון פנוי להקצאה ומחזירה את כתובתו כ- *void. כאשר הפונקציה אינה מצליחה לאתר שטח זיכרון פנוי, היא מחזירה ערך null.



נקודת ציון!

void יכול להיות כתובת לכל טיפוס, ולכן יש לבצע **הסבה בכוח** (casting) לסוג הטיפוס שברצוננו להחזיר.*

דוגמה

```
char *c_ptr, s_user[] = "AVI";
int i_ptr;

c_ptr = (char *) malloc (sizeof(s_user) +1);
if(c_ptr ! = null)
  printf("memory OK!);
else
  printf("No memory");
i_ptr = (int *) malloc(sizeof (int));
if(i_ptr ! = null)
  printf("memory OK!);
else
  printf("No memory");
```

הפונקציה calloc

הפונקציה **calloc** - הקצאת זיכרון ואפס את תוכנו.

מבנה הפונקציה:

```
void *calloc(size_t nitems, size_t size);
```

nitems מספר בתים להקצאה.

size_t קבוע של בורלנד.

size גודל הטיפוס להקצאה.

דוגמה לשימוש

```
char *str = NULL;

str = (char *) calloc(10, sizeof(char));
```

הקצאת מקום למערך מחרוזות בגודל 10.

שים לב!

*הפונקציה מקבלת מספר בתים לטיפוס ומקצה מקום בהתאם: מספר בתים*הטיפוס.*

הפונקציה מחזירה null אם אינה מצליחה לאתר שטח זיכרון פנוי.

דוגמה נוספת לשימוש

```
int *i_ptr
i_ptr = (int *) calloc (10, sizeof(int));
```

הפונקציה free

הפונקציה **free** - שחרור בלוק מוקצה בזיכרון. מבנה הפונקציה:

```
void free (void *block);
```

block* מצביע לבלוק אשר הוקצה בזיכרון.

הפונקציה משחררת אזורים בזיכרון שהוקצו על ידי הפונקציות malloc()‎ ו-calloc()‎.

שים לב!

***חובה** לשחרר בלוק שהוקצה בזיכרון לאחר סיום השימוש בו. התוצאות הנובעות ישירות מאי שחרור בלוק בזיכרון היא מילוי שטח הזיכרון, עד לנפילת התוכנית בגלל חוסר שטח בזיכרון.*

ניסיון לשחרר בלוק זיכרון שלא הוקצה, או במילים אחרות, קריאה לפונקציה free עם פרמטר שהוא כתובת כלשהי בזיכרון תביא לתוצאות בלתי צפויות.

דוגמה לשימוש

```
char *str;
str = (char *) malloc(10);              /* הקצה זיכרון לפי תווים */
strcpy (str, "Hod_Ami");        /* העבר מחרוזת לתוך האזור המוקצה */
.free(str);                        /* שחרר את שטח הזיכרון אשר הוקצה */
```

התוכנית הבאה מדגימה את תהליך הקצאה ושחרור זיכרון.

```
        /*****     chp1406.c  :קובץ תקליטור       *****/
                                              /* הקצאת זיכרון דינמית */
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <string.h>

void a_malloc(void);
void a_calloc(void);

/* -----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ---------------------------------------------------------*/
int main(void)
 {
  clrscr();
  a_malloc();
  a_calloc();
  printf("\nPress any key\n");
  getch();
  return(0);
 }
/* -----------------------------------------------------------
 ;  F U N C T I O N S
 ; -------------------------------------------------------- */
void a_malloc(void)
 {
  char *s_user;
                                /* הקצה זיכרון למחרוזת באורך 10 בתים */
  if ((s_user = (char *) malloc(10)) == NULL)
    {
        printf("Not enough memory\n");
        exit(1);
    }
  strcpy(s_user, "Learn C");            /* העבר תוכן למחרוזת */
                                          /* הקרן את המחרוזת על המסך */
  printf("(malloc) The string user is : %s\n", s_user);
```

```
  free(s_user);                    /*  שחרר את הזיכרון אשר הוקצה  */
 }

void a_calloc(void)
 {
  char *s_user = NULL;
                                        /*  הקצה זיכרון למחרוזת באורך 10 בתים  */
  if ((s_user = (char *) calloc(10, sizeof(char))) ==
                                                     NULL)
    {
        printf("Not enough memory\n");
        exit(1);
    }
  strcpy(s_user, "Learn C");              /*  העבר תוכן למחרוזת  */
                                              /*  הקרן את המחרוזת על המסך  */
  printf("(calloc) The string user is : %s\n", s_user);
  free(s_user);                      /*  שחרר את הזיכרון אשר הוקצה  */
 }
```

תוצאות ריצת התוכנית:

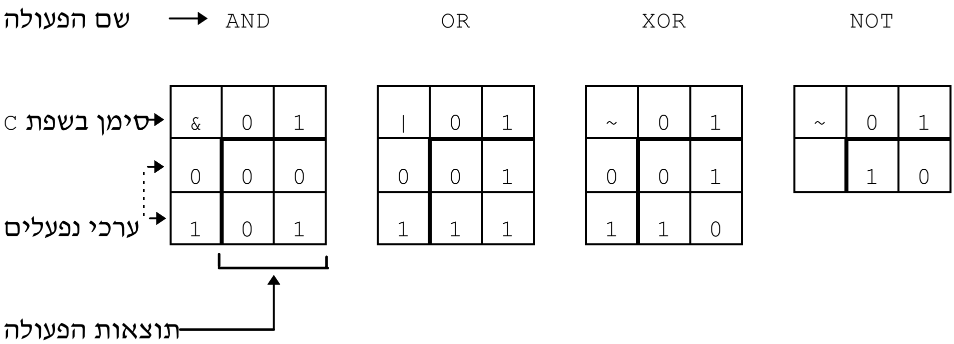
```
(malloc) The string user is : Learn C
(calloc) The string user is : Learn C

Press any key
```

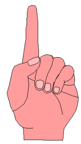
עבודה בסיביות

עבודה בסיביות מיועדת בעיקר למשתמשים בשפת C ביישומי זמן אמת וגרפיקה, בהם הגישה לסיביות היא הכרחית. אחת התכונות החשובות של שפת C היא היכולת לגשת ליחידת המידע הקטנה ביותר במחשב - אל ה**סיבית** (bit). עד כה עסקנו בבתים ופעולות בין משתנים שגודלם נע מבית אחד עד שמונה בתים. הסיבה לכך פשוטה: הטיפוס הקטן ביותר בשפת C הוא התו, שגודלו בית אחד. עובדה זו נובעת ממבנה הזיכרון במחשב, המאפשר גישה ישירה ליחידה של בית, וכתובת הזיכרון היא כתובת של בית מסוים. כעת נראה כיצד ניתן בעזרת ארבע פעולות בסיסיות להתייחס לסיביות בודדות ולבצע בעזרתן פעולות לוגיות.

לפניך טבלאות המתארות את ארבע הפעולות וערכיהן הלוגיים, כתלות באופרטורים של כל אחת מהן.



כפי שניתן לראות, קיים קשר בין הפעולות הלוגיות לפעולות המקבילות בסיביות. מכך נובע לדוגמה, הסימן '&&' לעומת הסימן הבודד '&'.



שים לב!

טעות נפוצה היא ל"שכוח" את אחד הסימנים כאשר עוסקים בפעולות לוגיות, ומקבלים תוצאות לא צפויות בזמן הרצת התוכנית (מהדר "אינטליגנטי" יעיר על טעות זו).

בניגוד לפעולות הלוגיות שהכרנו קודם, פעולות אלו מתבצעות על כל סיבית של האופרטורים **בנפרד**, והתוצאה המתקבלת היא אוסף התוצאות של הפעולות הבודדות.

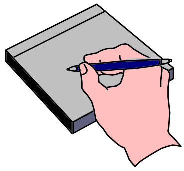
דוגמה

```
(0010)2 & (0110)2
0 and 0 = 0
0 and 1 = 0
1 and 1 = 1
1 and 0 = 0
```

תוצאת הפעולה: $(0010)_2$.

מכיון שבשפת C ניתן לייצג מספרים בבסיסים שונים, כל הפעולות הבאות הן חוקיות:

```
0xF8 & 0x15
0x19 ^ 037
   7 ^ 9
~((x&y) | c^d))
```



הערות!

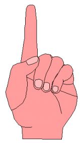
א. אין לבצע פעולות בסיביות בין מספרים ממשיים.

ב. מומלץ להשתמש בסוגריים, כדי להימנע מטעויות בהערכת הביטויים.

*ג. כדי לחשב את הערך של "**משלים ל-2**" (2's complement) ניתן לבצע **פעולות שלילה** (not) ולהוסיף 1 לתוצאה: ~x +1.*

דוגמה

```
x =        001101
~x    --> 110010
~x+1  --> 110011
```



שים לב!

בעזרת המשלים ל-2 מקובל להפוך מספר בינארי חיובי למספר בינארי שלילי ולהיפך.

שימושים נוספים

- **פעולת and** (&) משמשת לכיבוי סיביות לפי מסכות, וגם לבדיקת זוגיות של מספר (כל סיבית 0 במסכה מכבה סיבית בתוצאה).
- **פעולת OR** (|) משמשת להדלקת סיביות לפי מסכות (כל סיבית 1 במסכה מדליקה סיבית בתוצאה).
- **פעולת XOR** (^) משמשת לאיפוס משתנים, החלפת תוכן בין שני משתנים, וגם להפיכת סיביות מסוימות ובדיקת סיביות לפי מסיכה (כל סיבית 1 במסכה נבדקת בתוצאה).

דוגמאות

בדיקת הסיבית הימנית:
```
x & 1;
```

כיבוי הסיבית הימנית:
```
x & = ~1;
```

איפוס משתנה:
```
x = x^x;
```

בדיקת אי-זוגיות של מספר:
```
if(x & 1)
```

פעולות הזזה בסיביות

לאלה המכירים את שפת אסמבלר ועיבודי זמן אמת, ידועה חשיבות **פעולות הזזה** (shift). פעולות אלו מאפשרות כפל וחילוק מהירים בחזקות של 2, והזזה של סיביות במטרה לבצע עליהן בדיקות.

שים לב!

המהירות חשובה ביישומים שבהם זמן הביצוע של מחזור העיבוד הוא גורם קריטי.

רוב שפות המכונה מכילות ביטויים כגון:

```
ASL - Arithmetic Shift Left
ASR - Arithmetic Shift Right
```

בשפת C מציינים פעולות אלו על ידי הסימנים הבאים:

>> הזזת סיבית ימינה.

<< הזזת סיבית שמאלה.

לדוגמה,

```
0x01 << 2        התוצאה: 0x04
0x14 >> 1        התוצאה: 0x0A
```

אם נסתכל על הפרישה הבינארית של פעולות אלו, נבין את התוצאה:

$(00000100)_2$ <-- $(00000001)_2$ <<2

שים לב!

בעת הזזה שמאלה "נכנסים" מימין אפסים.

בפעולה זו "הזזנו" את הסיביות 2 מקומות שמאלה ועל ידי כך כפלנו את המספר ב-2^2 (כל הזזה הינה כפל ב-2').

שים לב!

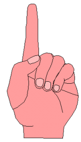
יש לזכור את המרכיבים של בית בזיכרון:

1	2	3	4	5	6	7	8
0	0	0	0	0	1	0	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

שים לב!

בכל פעולת הזזה שמאלה, הולכת לאיבוד הסיבית השמאלית ביותר. מסיבה זו, מותר לכפול מספר בחזקה של 2 על ידי הזזה שמאלה רק אם פעולה זו אינה גורמת לאובדן הסיבית בעל הערך 1 בקצה השמאלי ביותר.

בפעולה השנייה: $(00010100)_2 >> 1 \rightarrow (00001010)_2$ מזיזים את הסיביות מקום אחד ימינה.



שים לב!

משמאל נכנסים אפסים.

והתוצאה היא חלוקה ב-2 ($(14)_{16} = (20)_{10}$, $(A)_{16} = (10)_{10}$).

שפת C מתייחסת בצורה שונה לפעולות הזזה של משתנים וקבועים בעלי סימן וחסרי סימן. ערכים שליליים מיוצגים במחשב בשיטת משלים ל-2. שיטה זו מכתיבה שהסיבית השמאלית ביותר היא 0 עבור ערך חיובי, ו-1 עבור ערך שלילי.

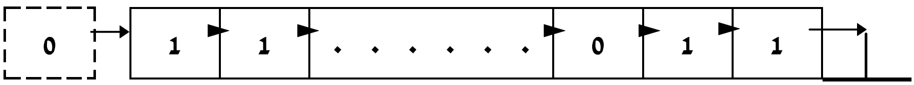
בפעולות הזזה ימינה של ערכים בעלי סימן, נשמר הערך של סיבית זו על מנת לשמור על הסימן. פעולות הזזה על ערכים בעלי סימן נקראות **הזזות אריתמטיות**, שכן הן משמרות את הסימן האריתמטי.

ניתן לראות סכימטית את פעולות ההזזה ימינה כך:



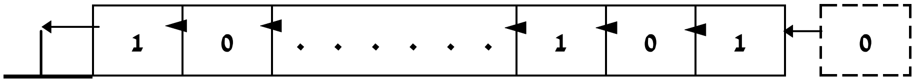
תרשים 14.4: *הזזה אריתמטית ימינה*

בפעולות הזזה ימינה של ערכים חסרי סימן, אין שימור סימן ותמיד ייכנס אפס ('0') משמאל. פעולות הזזה של ערכים חסרי סימן נקראות **הזזות לוגיות**. סכימטית נציג זאת בדרך זו:

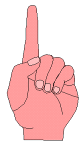


תרשים 14.5: *הזזה לוגית ימינה*

בשני סוגי ההזזות שמאלה נכנס אפס ('0') מימין והסיבית השמאלית הולכת לאיבוד:



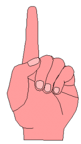
***תרשים 14.6:* הזזה שמאלה**



שים לב!

*בעת הזזה ימינה של **משתנה בעל סימן** (signed) יכולים להיות שני מצבים: כאשר נכנסת סיבית הסימן בפעולת ההזזה, נקרא לפעולה **Arithmetic shift**; וכאשר ייכנסו אפסים בפעולות ההזזה, נקרא לפעולה **Logical shift**.*

במשתנה (unsigned) ייכנסו אפסים בפעולות הזזה.



שים לב!

*כאשר לפנינו **ערך מסומן** (Signed value) ראינו שתחום טיפוס תו במקרה כזה הוא מ-128- עד 127, או 10000001 עד 01111111. כלומר, כדי לעבור ממספר חיובי לשלילי יש להפוך את ערכי הסיביות ולהוסיף 1.*

נבצע הזזה כפולה למספר $(63)_{10}$ = 00111111 נקבל את התוצאה 11111100, שערכה 4-. נציג זאת בדוגמת התכנות הבאה:

```
char num;

num = 63;
num = num << 2;
printf("%d", num);
```

```
-4
```

בתוכנית הבאה נלמד כיצד להפוך מספר integer לייצוג הבינארי שלו בעזרת פעולות על סיביות.

```
/*****        chp1407.c  :קובץ תקליטור        *****/

/*  קלוט ערך שלם והצג את הסיביות בעזרת אופרטורים לעבודה בסיביות  */

#include <stdio.h>
#include <conio.h>
void bit_display(int num);
```

```
/* -----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ---------------------------------------------------------*/
int main(void)
 {
  int num = 1;

  clrscr();
  printf("Display integer bits\n");
  while (num)
     {
      printf("\nEnter a number to convert <0> to quit : ");
      scanf("%d", &num);
      if (num)
      bit_display(num);
     }
  return(0);
 }
/* -----------------------------------------------------------
 ;  F U N C T I O N S
 ; ---------------------------------------------------------*/
void bit_display(int num)
 {
  int mask = 1, i;
  char bits[16];

  for (i = 0;i < 16;i++)
     {                                    /*  פירוק המספר לסיביות  */
      if (num & mask)          /*   0 כל מה ש- 1 נשאר 1 וכן 0 נשאר  */
         bits[15 - i] = '1';
      else
         bits[15 - i] = '0';
                   /*  הזזה של ספרה אחת אחורה לצורך בדיקת הסיבית הבאה  */
      mask <<= 1;
     }
  printf(" FEDC BA98 7654 3210\n");
  for (i = 0;i < 16;i++)
     {
      if(i % 4 == 0) printf(" ");
      printf("%c", bits[i]);
     }
 }
```

תוצאות ריצת התוכנית:

```
Display integer bits

Enter a number to convert <0> to quit :  12

FEDC BA98 7654 3210
0000 0000 0000 1100

Enter a number to convert <0> to quit : 0
```

תרגילים

אין תשובות בתקליטור.

1. מה היא התוצאה הבינארית של הפעולות הבאות:
   א. 01101001 & 11101011
   ב. 01110010 ¦ 00010010
   ג. 1001001 ^ 00001001
   ד. ~00101010 ¦ 010000101
   ה. (0xF8EA & 0x0801) ¦ 0x0F
   ו. ~((0x78 ^ 0x01) & (0x7F & 0x75))
2. כיצד תבדוק אם הערך של הסיבית החמישית מימין במספר 0x75 הוא 1? פרט את הפעולות שתבצע והביטוי הלוגי המתאים.
3. כיצד תהפוך ל- 0 ("תכבה") את הסיבית השמינית במספר 0x0397?
4. פרט שתי דרכים להפוך את ערכי הסיביות של המספר 0x35.
5. מה תהינה התוצאות של הביטויים הבאים:
   א. (0x35 & 0x07) >> 2
   ב. 0x15 ^ (0x05 <<3)
   ג. (0x15 << 12)

פרק 15

רקורסיה

רקורסיה (Recursion) מתקיימת כאשר פונקציה כלשהי קוראת לעצמה, באופן ישיר או עקיף. **קריאה ישירה** הינה קריאה של הפונקציה אל עצמה בתוך אחת משורות הפקודות שלה. **קריאה עקיפה** נעשית כאשר פונקציה א' קוראת לפונקציה ב', וזו קוראת שוב לפונקציה א'. שפת C מאפשרת לכל פונקציה להיקרא באופן רקורסיבי, דהיינו: **להיקרא על ידי עצמה**.

באמצעות הרקורסיה ניתן לכתוב תוכניות המבצעות משימות מורכבות, באופן פשוט ונוח לקריאה ולהבנה. נדגיש עם זאת, שהרקורסיה אינה חוסכת זיכרון בזמן הריצה, מכיון שהיא נתמכת על ידי מנגנון הצורך שטח זיכרון גדול. תהליך הבנת הרקורסיה ודרך השימוש בה אינו פשוט, ולכן נציג את הנושא במפורט ובשלבים.

הפעלה של פונקציה רקורסיבית

בדוגמאות הבאות נלמד איך פועלת הרקורסיה ואיך מתנהגת פונקציה רקורסיבית.

התבונן בקטע התוכנית הבא:

```
/*****     קובץ תקליטור: chp1501.c     *****/

#include <stdio.h>

void star(void)
{
   printf("*") ;
   star() ;
}

void main(void)
{
   star() ;
}
```

בדוגמה זו, התוכנית הראשית קוראת לפונקציה ()star, אשר מדפיסה כוכבית וקוראת לעצמה שוב. לכן, ()star הינה פונקציה רקורסיבית. הקריאה הנוספת ל-()star, תגרום להדפסת כוכבית נוספת וכן הלאה. התהליך הזה יימשך לעד, שכן כל קריאה ל-()star מדפיסה כוכבית וקוראת שוב ל-()star. המסך יתמלא ברצף בלתי פוסק של כוכביות והתופעה תזכיר לולאה אינסופית.

התבונן בקטע תוכנית נוסף להגדרת ()star:

```
         /*****      chp1502.c  :קובץ תקליטור       *****/
#include <stdio.h>
void main(void)
{
   star() ;
}
     /*  הפונקציה עצמה  */
void star(void)
{
   star() ;
   printf("*") ;
}
```

ההבדל מתבטא כאן בשינוי מיקום השורות: הקריאה הרקורסיבית ל-()star מתבצעת **לפני** הדפסת הכוכבית. כדי לנתח את תוצאת ההרצה, נתחיל בקריאה הראשונה ל-()star מהתוכנית הראשית. הפעולה הראשונה ש-()star מבצעת היא קריאה לעצמה, ולכן ()star תופעל פעם נוספת. פעולה זו תגרום לקריאה ל-()star וכך הלאה: תתקבל סדרת הפעלות בלתי פוסקות של הפונקציה ()star בזו אחר זו, ולפקודת ההדפסה לא נגיע לעולם. השינוי הקטן שביצענו גרם לכך שמטרתנו האמיתית - הדפסת רצף של כוכביות - לא תושג.

בעת הפעלת התוכנית נדמה לנו שהמחשב "תקוע" מכיון שלא מתבצעת כל פעולת "פלט", ואמנם כך הדבר: המחשב עסוק בקריאות חוזרות ונשנות לפונקציה ()star. פקודת ההדפסה אינה מתבצעת, כיון שאין חוזרים מהפונקציה ()star. כזכור, חזרה מפונקציה מתבצעת על ידי הפקודה return, או כאשר מגיעים לסוף הפונקציה. במקרה זה, אף לא אחד מהדברים האלה מתבצע ב-()star. נוכל ל"תקן" את ()star, כדי שנוכל לצאת מהפונקציה. נעשה זאת על ידי התניית ההפעלה הרקורסיבית מתוך ()star:

```
         /*****      chp1503.c  :קובץ תקליטור       *****/
#include <stdio.h>
     /*  הפונקציה עצמה  */
void star(int i)
{
   if (i>1)
      star(i-1) ;

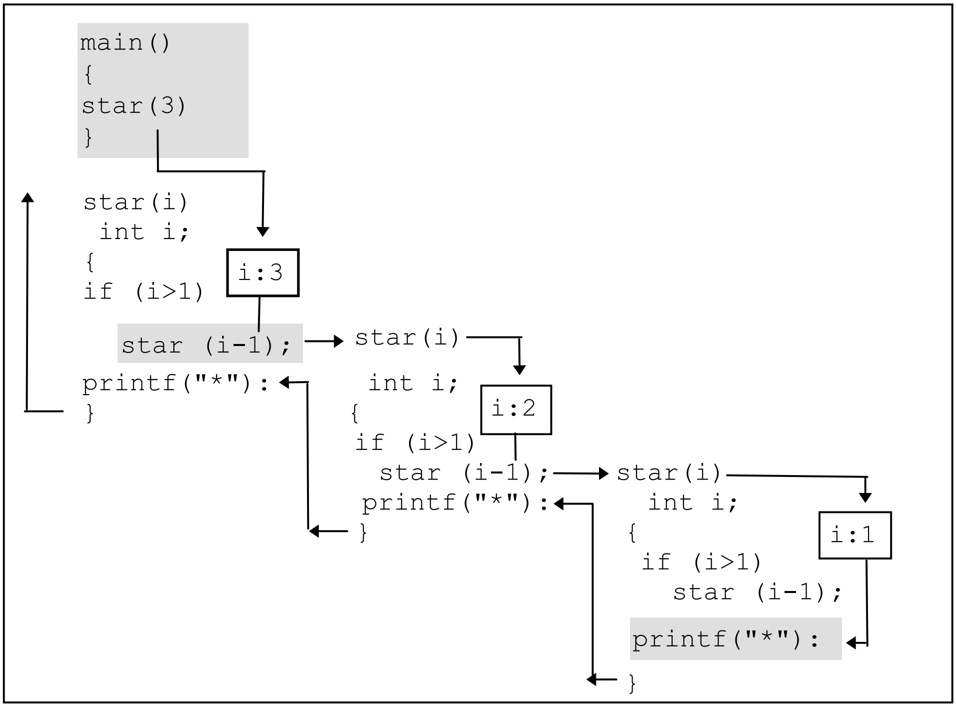
   printf("*") ;
}
```

בנוסח זה של star()‎ הוספנו משתנה אוטומטי i, שערכו קובע אם תבוצע קריאה רקורסיבית ל-star()‎, או לא. משתנה זה מוקצה מחדש בכל פעם שקוראים ל-star()‎. אם i גדול מ-1, מתבצעת קריאה נוספת ל-star()‎ עם הערך i-1. ערך זה יגיע ל-i של star()‎ ה"חדש". נדגים תהליך זה באמצעות התוכנית הראשית הבאה, ובאמצעות תרשים 15.1:

```
main()
{
   star(3);
}
```

ההפעלה הראשונה של star()‎ מקצה משתנה אוטומטי i בעל ערך 3. כל הפעלה נוספת של star()‎ מעבירה את הערך של i-1, ולכן תהיינה שתי קריאות נוספות ל-star()‎ עם ערכים 2 ו-1 בהתאמה. בקריאה השלישית, בה i=1, לא תתבצע קריאה רקורסיבית נוספת, כיון שהתנאי שהצבנו אינו מתקיים.

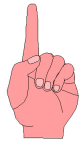
בנקודה זו תתבצע הפקודה printf()‎ ותודפס הכוכבית ("*") בפעם הראשונה, ותתבצע חזרה מהפונקציה (כי הגענו לסופה). בקרת התוכנית תחזור לפונקציה הקוראת, שהיא ההפעלה הקודמת של star()‎. נקודת החזרה מהפונקציה היא פקודה אחת לאחר הקריאה עצמה, ולכן תתבצע פקודת ההדפסה השנייה ושוב תהיה חזרה מהפונקציה. באופן דומה תתבצע פקודת ההדפסה השלישית וחזרה ל-main()‎.



תרשים 15.1: *סכימת תהליך הדפסה רקורסיבי*

הפלט על המסך ייראה כך:

```
***
```



שים לב!

*הכוכבית **הראשונה** המופיעה הודפסה על ידי הקריאה **השלישית** ל-()star!*

ניתוח דומה, אם כי ארוך יותר יראה, שקריאה ראשונית ל-()star בתוכנית הראשית הבאה, תפיק הדפסה רצופה של 10 כוכביות:

```
main()
{
   star(10);
}
```

```
**********
```

בדוגמאות הקודמות ביצענו לולאה באמצעות רקורסיה, ללא שימוש במשפטי הבקרה for, או while. הכוכבית ה**ראשונה** בשורת הכוכביות הודפסה על ידי הפקודה ()printf של ההפעלה ה**אחרונה** של ()star, והכוכבית ה**אחרונה** בשורה - על ידי ההפעלה ה**ראשונה** של הפונקציה.

ניתן להדגים את תהליך הביצוע של הפונקציה על ידי החלפה של הפקודה:

```
printf("*");
```

בפקודה

```
printf ("%d ",i);
```

התוכנית המתוקנת:

```
star(int i)
{
   if (i > 1)
      star (i-1)
   printf ("%d ", i);
}
```

הפעלת (10)star תפיק את הפלט הבא:

```
1 2 3 4 5 6 7 8 9 10
```

כדי לקבל סדר יורד של מספרים (שהוא הסדר הנכון של הקריאות) נוכל להחליף את מיקום הפקודות בפונקציה ()star:

```
star(int i)
{
   printf ("%d ", i);
   if (i>1)
      star (i-1)
}
```

ההדפסה מתבצעת לפני הקריאה הרקורסיבית. הפעלת (10)star תפיק את הפלט הבא:

```
10 9 8 7 6 5 4 3 2 1
```

כל קריאה רקורסיבית ל-star() הינה כניסה **לדרגת קינון** (Nesting) נוספת של הרקורסיה: בדוגמה star(3) נכנסנו לדרגת קינון 3 ובדוגמה star(10) הגענו לדרגה 10. התנאי if (i>1) נקרא "**תנאי עצירת הרקורסיה**", כי על פי ערכו הלוגי מחליטים אם להמשיך בקריאות רקורסיביות או לא. כדי שרקורסיה תהיה סופית, היא חייבת להכיל תנאי עצירה כלשהו, שיפסיק את תהליך הקריאה בשלב מסוים ויאפשר חזרה אל התוכנית הקוראת.

מנגנון הרקורסיה

עד כה התעלמנו מהמתרחש במחשב בזמן הביצוע של פונקציה רקורסיבית. ייתכן שהתרשימים הסכימטיים שהראינו במהלך הפרק יצרו את האשלייה שבכל קריאה רקורסיבית לפונקציה היא נפרשת מחדש בזיכרון המחשב. הדבר אינו כך! למעשה, במחשב קיים רק עותק אחד של הפונקציה, אשר מבוצע מחדש בכל פעם שיש קריאה רקורסיבית.

מנגנון הרקורסיה אחראי לעקוב אחד הנקודה בה נפסק הביצוע הקודם של הפונקציה, וכשתסתיים הקריאה החדשה ותהיה חזרה מהפונקציה, הוא ידאג לחזור לנקודה זו ולהמשיך בביצוע. לעומת זאת, בכל הפעלה רקורסיבית של הפונקציה מוקצים המשתנים האוטומטיים והערכים הפורמליים שלה מחדש, והם ייחודיים לאותה הפעלה. אם ברצוננו להתייחס **לאותו משתנה** בכל אחת מהקריאות הרקורסיביות, עלינו להגדירו כמשתנה גלובלי או סטטי.

ערך מוחזר מפונקציה רקורסיבית

דוגמה קלאסית לפונקציה רקורסיבית היא פונקציה המחשבת עצרת. עצרת של המספר השלם n מחושבת כמכפלת כל השלמים שקטנים, או שווים ל-n:

```
n! = 1*2*3*...*(n-1)*n
```

לפניך הגדרה של הפונקציה fact(), המבצעת את סדרת המכפלות הדרושות בצורה רקורסיבית:

```
      /*****       chp1504.c :קובץ תקליטור       *****/

     #include <stdio.h>

(1)  fact(int n)
(2)
(3)  {
(4)    if (n>1)
(5)      return(n * fact (n-1)) ;
(6)    else
(7)      return(1);
(8)  }
```

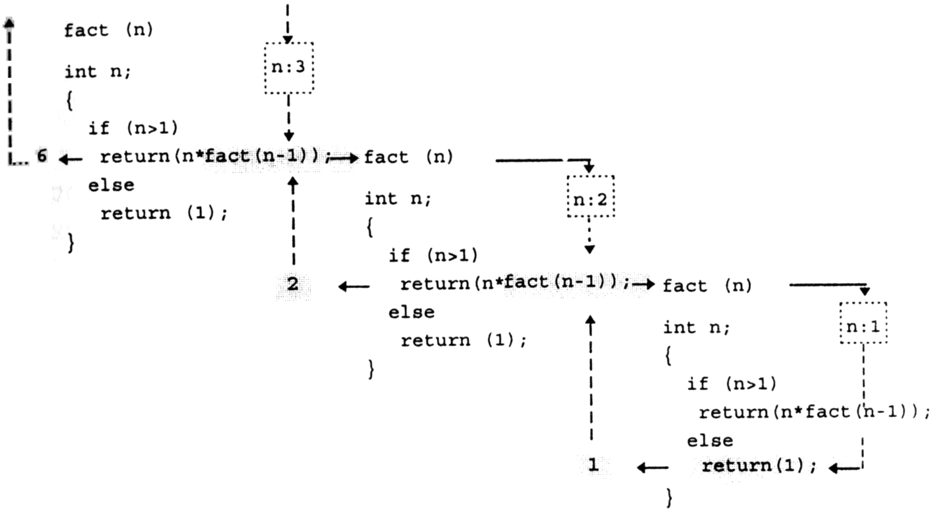
הסבר:

א. הקריאה הרקורסיבית מבוצעת כאן בתוך הפקודה return שבשורה 5.

ב. הפקודה if בשורה 4, היא תנאי עצירת הרקורסיה. כאשר n==1 תתבצע הפקודה return(1), ולא תתבצע קריאה רקורסיבית.

ג. כאשר (n > 1) מוחזרת התוצאה n*fact(n-1), אשר מבוססת על קריאה רקורסיבית ל-fact().

נדגים זאת שוב באופן סכימטי בתרשים 15.2, עבור ההפעלה fact(3).



*תרשים 15.2 : **1, 2 ו-6 הם הערכים המוחזרים על ידי הפונקציה fact()***

נפעיל את התוכנית:

```
      /*****      chp1504.c  :מתוך קובץ תקליטור          *****/

main()
{
  printf("%d! = %d\n", 5, fact(5)) ;
}
```

הפלט:

```
5! = 120
```

הבסיס המתמטי לחישוב !n באמצעות רקורסיה הינו השוויון הבא:

```
n!=1*2*3...*(n-1)*n = (1*2*3...*(n-1))*n = (n-1)!*n
```

כלומר, כדי לחשב !n, נחשב תחילה את !(n-1) ואת התוצאה נכפול ב-n.

נראה בהמשך, שהבסיס לפונקציות רקורסיביות הינו היכולת לנסח את אופי הבעיה בשני שלבים:

א. פתרון בעיה דומה בעלת דרגת קושי אחת פחות.

ב. שילוב צעד נוסף לקבלת הפתרון הנדרש.

את בעיית העצרת ננסח כך:

א. פתרון !(n-1).

ב. נכפול את התוצאה של שלב א׳ ב-n ונקבל !n.

תרגיל:

כתוב פונקציה רקורסיבית ()mult, המקבלת שני ערכים וכופלת אחד בשני, על בסיס השוויון המתמטי הבא:

```
x * y = (x * (y-1)) + x
```

עקרונות תכנון של פונקציות רקורסיביות

כפי שראינו בסעיפים הקודמים, קיימים כמה עקרונות עליהם יש להקפיד כדי לקבל פונקציות רקורסיביות סופיות, המחשבות תוצאה מסוימת, או מבצעות פעולה כלשהי על פי עקרונות אלה.

נגדיר שיטה לתכנון פונקציה רקורסיבית:

א. הגדר את הבעיה הכללית שברצונך לפתור.

ב. חלק את הבעיה לשני שלבים:

1. פתרון בעיה דומה בעלת דרגת קושי אחת פחות.
2. הגדרת השלב הנוסף הדרוש לפתרון הבעיה הכללית, בהנחה ששלב 1 ניתן לפתרון.

ג. קבע מהו התנאי לעצירת הרקורסיה.

ד. כתוב את הפונקציה על פי השלבים הבאים:

1. התחל בבדיקת התנאי לעצירת הרקורסיה. אם התנאי מתקיים בצע חזרה מהפונקציה, עם או בלי פעולת חישוב.
2. כתוב את שלב ב(2) מתוך הנחה ששלב ב(1) פועל כנדרש.

זוהי נקודה חשובה, ולכן נפרט את הפעולה:

אנו מניחים מראש, ללא בדיקה, כי הפונקציה שאנו כותבים "מסוגלת" לתת פתרון בעל דרגת קושי אחת פחות, ועל פי זה אנו כותבים את השלב הנוסף ב(2). חשוב ביותר בשלב התכנון והכתיבה של הפונקציה "לתת אמון" ביכולתה לבצע את ב(1), מבלי לבחון כיצד הדבר נעשה. סוד ההצלחה של פונקציה רקורסיבית טמון בחלוקה הנכונה של הבעיה לשלבים ב(1) ו-ב(2).

ה. בחן באופן ידני את הפונקציה שכתבת. עשה זאת באמצעות דוגמאות פשוטות של הבעיה הכללית (כפי שעשינו בתכנות הפונקציות star(3) ו-fact(3)). סביר להניח שבשלב זה תגלה שגיאות אפשריות, או שתוכיח את נכונות הפתרון.

נדגים כעת מספר בעיות ופתרונן באופן רקורסיבי, על פי השלבים שתיארנו לעיל. קיימות שיטות נוספות לתכנון פונקציות רקורסיביות באמצעות "קיצורי דרך" ותחבולות, אך לא נעסוק בכך. אנו ממליצים לנקוט בשיטה שהצגנו, לפחות עד לקבלת מיומנות בעבודה עם פונקציות מסוג זה.

בעיות ופתרונן ברקורסיה

בעיה 1

א. הגדרת הבעיה: היפוך של מחרוזת בעלת n תווים.

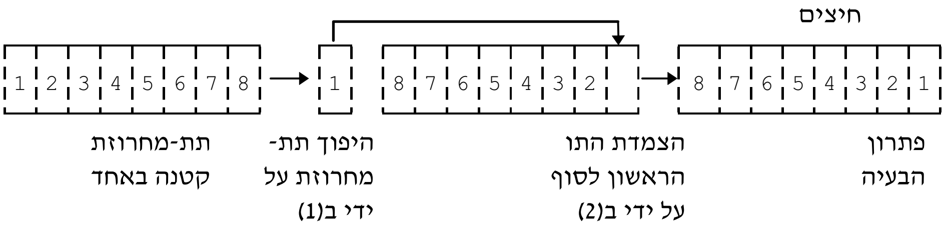
דוגמה:

```
"ABCD E" -> "E DCBA"
"THIS IS A STRING" -> "GNIRTS A SI SIHT"
```

ב. חלוקת הבעיה לשני שלבים:

1. היפוך של מחרוזת בעלת תו אחד פחות מהמחרוזת המבוקשת (n-1 תווים).
2. הצמדת התו הראשון במחרוזת לסוף תת-המחרוזת המתקבלת בשלב ב(1). אנו "מאמינים" כאן, שתת-המחרוזת "תטופל" כראוי על ידי שלב ב(1).

פעולות אלו מוצגות באופן סכימטי בתרשים 15.3.



*תרשים 15.3: **תהליך רקורסיבי להיפוך מחרוזת***

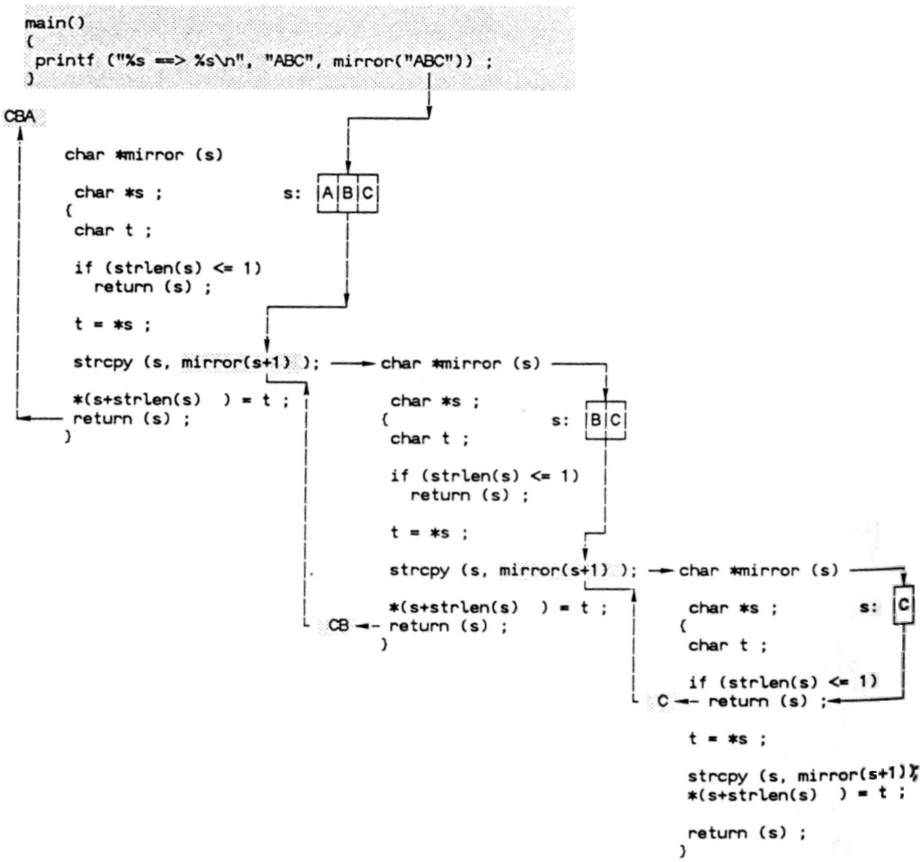
ג. תנאי עצירת הרקורסיה: כאשר אורך המחרוזת הוא 1 - אין משמעות לחלוקה של סעיף ב'. לכן נעצור את הרקורסיה ונחזיר את התו עצמו. היפוך של מחרוזת בעלת תו אחד בודד הינה המחרוזת המקורית ללא שינוי.

ד. תוכנית הפונקציה:

```
/*****    chp1505.c  קובץ תקליטור:    *****/
#include <stdio.h>
char *mirror (char *s)
{
   char t ;
   if (strlen(s) <= 1)
      return(s);                          /*  עצירת הרקורסיה  */
   t = *s ;
   strcpy (s, mirror(s+1));               /*  קריאה רקורסיבית  */
   *(s+strlen(s)) = t ;
   return(s) ;
}
```

ה. בחינת הפונקציה עבור מחרוזת פשוטה "ABC".

נכתוב תוכנית ()main הקוראת לפונקציה עם הפרמטר "ABC". את מהלך הפתרון נציג באופן סכימטי בתרשים 15.4.



*תרשים 15.4 : **מהלך פתרון רקורסיבי להיפוך מחרוזת***

```
ABC -->  CBA
```
תוצאת ההרצה היא:

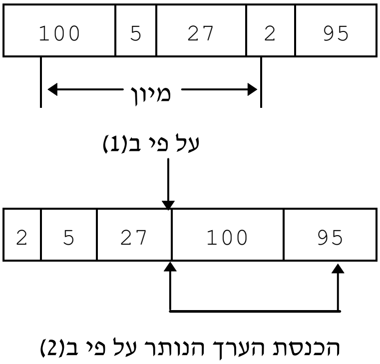
בעיה 2

א. הגדרת הבעיה: מיון מערך בעל n ערכים שלמים, לפי סדר מספרי עולה.

ב. חלוקת הבעיה לשני שלבים:

1. מיון מערך בעל (n-1) שלמים על פי סדר עולה.
2. הכנסת השלם הנותר (ה-n-י) למקום הנכון.

אנו "מאמינים" שהמערך הנותר על פי ב(1) כבר ממוין. תיאור סכימטי לפתרון זה ניתן בתרשים 15.5.



*תרשים 15.5: **סכימת מיון בשיטת רקורסיה***

ג. תנאי עצירת הרקורסיה: כאשר קיים במערך רק ערך בודד, אין משמעות לחלוקה על פי סעיף ב. במקרה זה המערך נשאר ללא שינוי.

ד. התוכנית:

```
        /*****     chp1506.c  :קובץ תקליטור      *****/
#include <stdio.h>
void sort (int nums[], int n)
{
  int i;
  if (n<=1)
    return ;

  sort (nums, n-1) ;                          /* קריאה רקורסיבית */
```

```
  while ((nums[n-1] < nums[n-2]) && (n > 1))
    {
        i = nums[n-1] ;                        /* החלפת ערכים */
        nums[n-1] = nums[n-2] ;
        nums[n-2] = i ;
        --n ;
    }
}
```

ה. בחינת הפונקציה עבור המערך הפשוט

2	1	3

תוכן []nums	הערך n		עומק הרקורסיה
2 1 3	3		1
2 1 3	2		2
2 1 3	1→	עצירת הרקורסיה וחזרה	3
2 3 1	2		2

הפקודה while גרמה להחלפה בין 3 ל-1.

3 2 1	3		1

הפקודה while גרמה להחלפה בין 2 ו-3 אך לא בין 2 ו-1.

חזרה לתוכנית הראשית: המערך ממוין! 0

שים לב עד כמה פשוטה תוכנית מיון בעזרת הרקורסיה.

תרגיל

כתוב פונקציה למיון מערך בסדר מספרי יורד.

עצים בינאריים

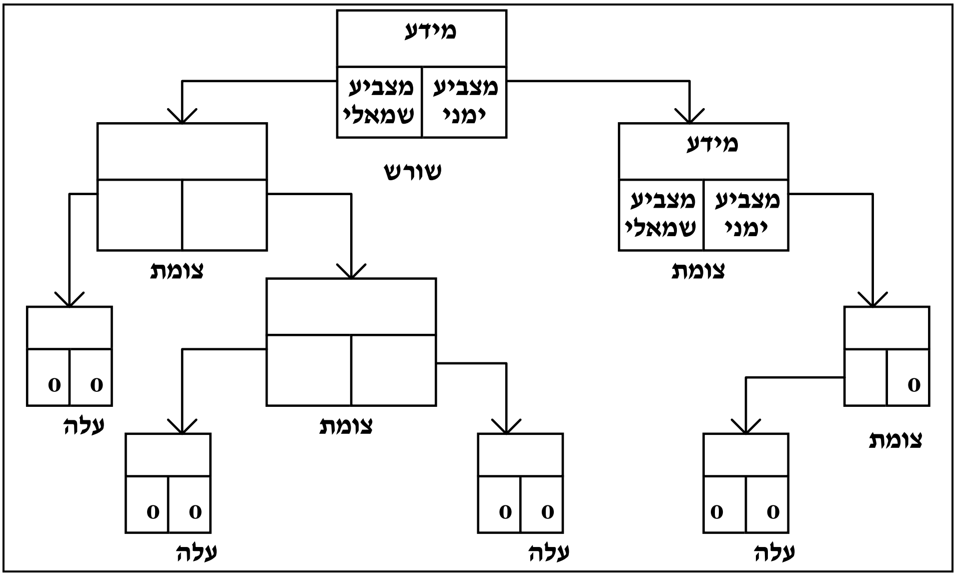
עץ בינארי (Binary Tree) הינו מבנה נתונים נפוץ ביישומי תוכנה רבים. הוא מורכב מ**שורש** בודד, **צמתים** (Nodes) **ועלים** (Leaves). ה**שורש** (Root) של העץ הבינארי הוא האיבר העליון ביותר שממנו "צומח" העץ, ועל כן הוא "מייצג" אותו. מן השורש ומן הצמתים יש התפצלויות נוספות, עד לעלים, שמהם אין כל התפצלויות שהן.

לשורש ולכל אחד מהצמתים יש שני מצביעים, ימני ושמאלי, המצביעים לצומת או לעלה. בנוסף למצביעים ניתן לאחסן בשורש, בצמתים ובעלים מידע כלשהו. זוג המצביעים הוא שנותן לקבוצה זו את שמה - **עץ בינארי**. המצביעים לעלים ולצמתים נקראים **ענפים**: ענף שמאלי וענף ימני - בהתאם למצביעים. בדרך כלל מכנים את האיברים בעץ הבינארי על פי "ייחוס משפחתי". כל צומת הוא אב (או "**צומת אב**") וכל איבר הנמצא בקצה אחד הענפים שלו נחשב לבנו - בן ימני או בן שמאלי בהתאמה. באותו הקשר, ניתן להגדיר סבים, נכדים וכו'.

בשפת C ניתן לייצג כל אחד מהאיברים האלה באמצעות מבנה הדומה למבנה של רשימה מקושרת בעלת זוג מצביעים.

לשורש ולצמתים ניתן "לקשור" צומת או עלה בודד. במקרה זה, המצביע הנותר מכיל 0 (null). עלה מוגדר "**כצומת מנוון**" אשר שני המצביעים בו מאופסים והוא מכיל רק את המידע הנוסף. מטבעו, עלה יכול להפוך לצומת על ידי שרשור עלה לאחד ממצביעיו, בדומה להוספת איבר לסוף רשימה מקושרת.

בתרשים 15.6 ניתן ייצוג סכימטי של עץ בינארי.



*תרשים 15.6: **סכימה של עץ בינארי***

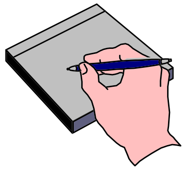
המבנה הפשוט ביותר של עץ בינארי מכיל פריט מידע בודד, מספר שלם למשל:

```
struct Btree {
                  int  info ;
                  struct Btree *left ;
                  struct Btree *right ;
             } root;
```

אחד השימושים לעץ בינארי הוא אחזקת מידע על פי סדר מיון כלשהו. העץ הבינארי מאפשר הוספת מידע לעץ ממוין ושליפתו בצורה נוחה ביותר. שיטת המיון מתבצעת כך:

א. התחל מהשורש. הבן השמאלי מכיל איבר קטן יותר והבן הימני מכיל איבר גדול יותר.

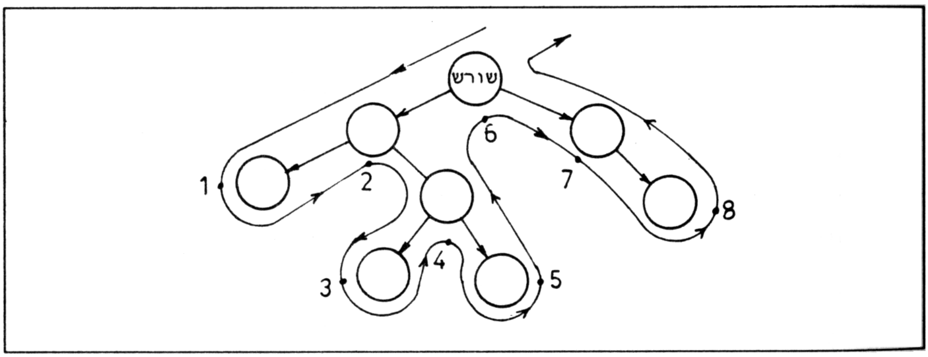
ב. עבור לאחד הבנים. בנו השמאלי של זה מכיל איבר קטן יותר והימני - איבר גדול יותר. המשך בדרך זו עד לעלים, אשר להם אין בנים.



הערה!

***בעץ בינארי ממוין**, נמצא האיבר הקטן ביותר בעלה השמאלי ביותר.*

נתבונן בעץ בינארי ממוין המוצג בתרשים 15.7 ונערוך מסלול "טיול" על פני ענפיו, כך שנעבור על פי סדר המיון על כל המידע המצוי בו.



תרשים 15.7 : ***סכימה של עץ בינארי ממוין***

אילו הדפסנו את המידע המוכל בכל צומת, על פי הסדר המסומן לאורך מסלול ה"טיול", היתה מתקבלת רשימה ממוינת של המידע (בהנחה שהעץ אכן ממוין בשיטה שתוארה קודם). טיול על פני עץ בינארי ממוין והדפסת המידע שבו על פי סדר המיון, הינו יישום קלאסי של פונקציה רקורסיבית, וקשה מאוד לבצע זאת ללא מנגנון כזה.



בעיה 3

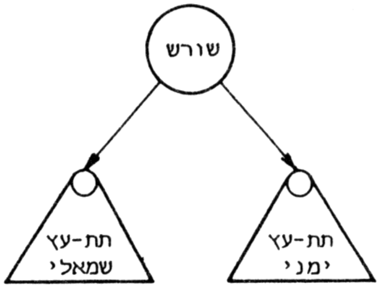
א. הגדרת הבעיה: נתון עץ בינארי ממוין המיוצג על ידי השורש, בעל איברים מטיפוס struct Btree. יש לכתוב פונקציה רקורסיבית להדפסת המידע שבו, על פי סדר המיון, תוך טיול על העץ.

ב. פירוק הבעיה לשלבים:

1. נייצג עץ בינארי ממוין כשורש, שאליו מקושרים מימין ומשמאל שני תתי-עצים בינאריים ממוינים. שורשיהם הינם הבן הימני והבן השמאלי של השורש המקורי. נכנה אותם תת-עץ ימני ותת-עץ שמאלי בהתאמה.

תיאור סכימטי של העץ ניתן בתרשים 15.8.

נניח, כי הפונקציה מסוגלת לבצע טיול והדפסה על פי המיון של כל תת-עץ. בכך נציג בעיה בעלת דרגת קושי אחת פחותה יותר. אפילו אם אחד מתת-העצים אינו קיים (כלומר לשורש המקורי יש בן יחיד), הרי שבתת-עץ הנותר יש צומת אחד פחות (השורש המקורי).



תרשים 15.8: ***סכימה כללית של עץ בינארי***

2. כדי לבצע את הפתרון הכללי נדפיס את תת-העץ השמאלי לפי סדר המיון על ידי קריאה רקורסיבית נוספת לפונקציה. אחר כך נדפיס את המידע בשורש, ונבצע קריאה רקורסיבית נוספת להדפסה של תת-העץ הימני לפי סדר המיון.

ג. תנאי עצירת הרקורסיה: כאשר אחד מהמצביעים של השורש מאופס, פירוש הדבר שאין מתחתיו תת-עץ, ולכן לא נבצע קריאה רקורסיבית לפונקציה. לגבי הפונקציה הרקורסיבית - השורש הוא השורש של תת-העץ שבו אנו עוסקים.

ד. תוכנית להדפסת עץ בינארי ממוין:

```
        /*****      chp1507.c :קובץ תקליטור        *****/

void p_sorted (struct Btree *root)
{
  if (root->left !=0)
    p_sorted (root->left) ;

  printf("%d ",root->info) ;

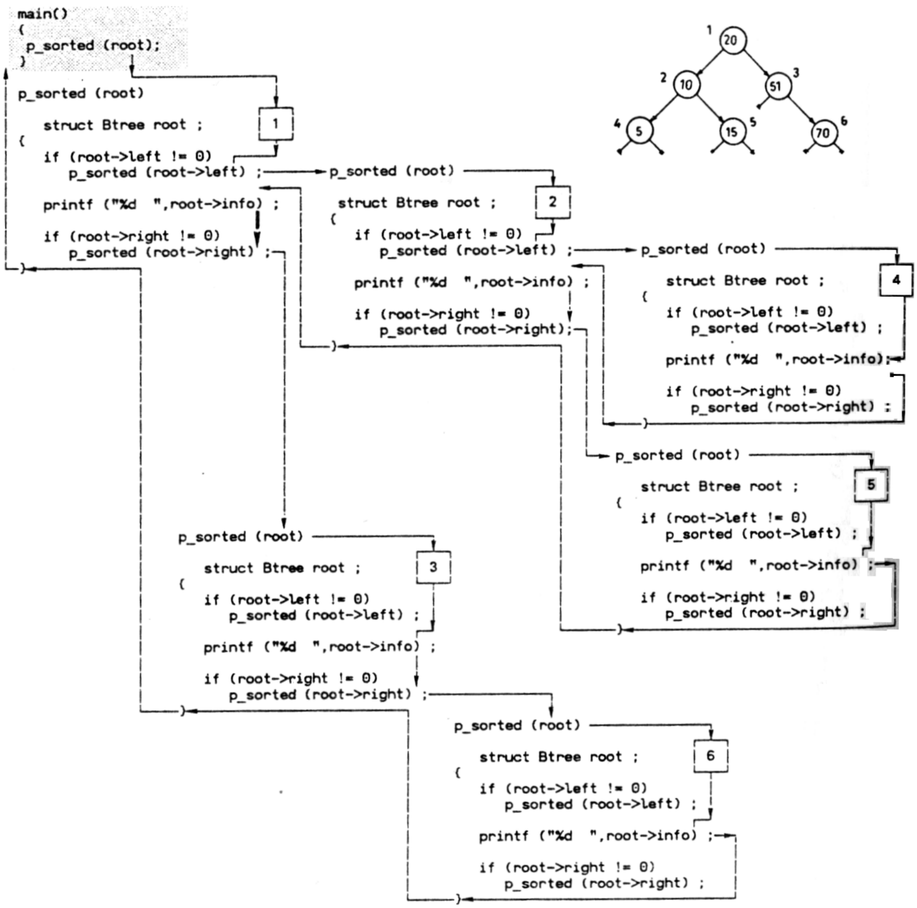
  if (root->right !=0)
    p_sorted (root->right) ;
}
```

פונקציה זו פשוטה ומובנת. שמרנו כאן על כל כללי התכנון של פונקציה רקורסיבית, ואנו מביאים לידי ניצול מלא את הפוטנציאל הרב הטמון במנגנון הרקורסיה.

תרגיל:

כתוב פונקציה רקורסיבית להדפסת המידע בעץ בינארי ממוין על פי סדר מיון יורד.

ה. נבחן את הפונקציה על פי הדוגמה הפשוטה המוצגת בתרשים 15.9. לשם נוחות, מספרנו את הצמתים במספרים שוטפים משמאל לימין, וציינו זאת גם בתוכנית.



*תרשים 15.9 : **סכימה להדפסת עץ בינארי***

פלט התוכנית יהיה:

```
5 10 15 20 51 70
```

מגדלי האנוי

לסיכום נציג משחק מעניין, הניתן לפתרון אלגנטי באמצעות פונקציה רקורסיבית. הסיפור הוא על משחק המכונה **"מגדלי האנוי"** (Towers of Hanoi).

המשחק מורכב משלוש יתדות, אשר על אחת מהן מונחות n דיסקיות בעלות קוטר הולך וקטן מלמטה כלפי מעלה ועל-כן יוצרות תבנית "מגדל". מטרת המשחק היא להעביר את n הדיסקיות ליתד אחר, כאשר הן מסודרות כמו בתחילה. להעברה זו אפשר להשתמש ביתד ביניים.

חוקי המשחק הם:

א. בכל צעד מותר להעביר דיסקית אחת עליונה בלבד מעמוד לעמוד.

ב. אין להניח דיסקית על גבי דיסקית קטנה ממנה.

נפתור בעיה זו בשלבים לפי הנוהל שלמדנו:

א. הגדרת הבעיה: קיימות שלוש יתדות שנסמן אותן במספרים 1, 2, 3. על יתד 1 מונחות דיסקיות בעלות קוטר הולך וקטן מלמטה למעלה. יש לכתוב פונקציה המדפיסה את סדרת הצעדים שיש לבצע בכדי להעביר, על פי חוקי המשחק, את מגדל הדיסקיות ליתד 3 באמצעות יתד הביניים 2.

ב. חלוקת הבעיה לשלבים:

1. נניח כי הפונקציה מסוגלת להעביר את n-1 הדיסקיות העליונות ליתד 2, באמצעות יתד 3. מכיון שהדיסקית התחתונה היא הגדולה ביותר, לא תפר פעולה זו את חוקי המשחק.

2. לאחר העברת n-1 הדיסקיות ליתד 2, נעביר את הדיסקית ה-n (הגדולה ביותר) מיתד 1 ליתד 3. באמצעות ב(1), נעביר את n-1 הדיסקיות מיתד 2 ליתד 3 באמצעות יתד 1.

ג. תנאי עצירת הרקורסיה: כאשר קיימת דיסקית בודדת, ניתן להעביר אותה ליתד היעד באופן ישיר, מבלי לפגוע בחוקי המשחק ובלי להזדקק לקריאה רקורסיבית.

ד. התוכנית:

נגדיר תחילה את הערכים הדרושים לפונקציה:

n מספר הדיסקיות ב"מגדל" המקורי

sp מספר היתד עליו מצוי המגדל בן n הדיסקיות

tp מספר יתד היעד להעברת מגדל הדיסקיות

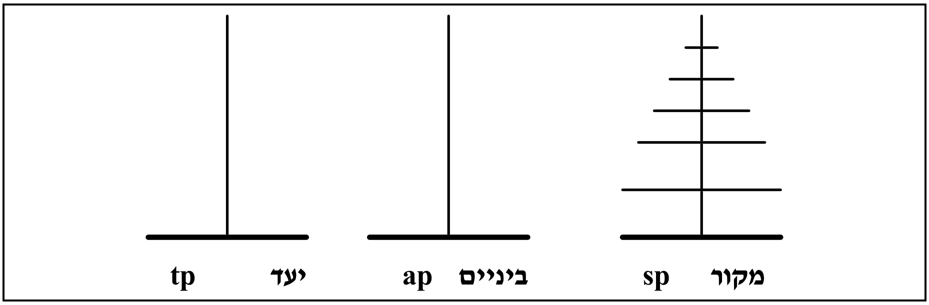
ap מספר יתד הביניים, שבאמצעותו מתבצעת ההעברה

```
          /*****      chp1508.c :קובץ תקליטור       *****/

hanoi (int n,int sp,int tp,int ap)

{
  if (n==1)
    printf("move disk from pole %d to pole %d/n",
                                        sp, tp) ;
  else
    {
    hanoi (n-1, sp, ap, tp) ;
    printf("move disk from pole %d to pole %d/n",
                                        sp, tp) ;
    hanoi (n-1, ap, tp, sp) ;
    }
}
```

בתרשים 15.10 ניתן תיאור הפעולה של הפונקציה.



*תרשים 15.10 : **סכימה של פתרון מגדלי האנוי***

להלן דוגמת הרצה של הפונקציה ()hanoi עבור 3 דיסקיות. לצורך הרצתה הכנו את התוכנית הראשית הבאה:

```
void main(void)
{
  printf("HANOI TOWERS with 3 DISKS: \n\n") ;
  hanoi (3, 1, 3, 2) ;
}
```

```
 HANOI TOWERS with   3 DISKS:

HANOI : n=3 sp=1 tp=3 ap=2
  HANOI : n=2 sp=1 tp=2 ap=3
    HANOI : n=1 sp=1 tp=3 ap=2
    move disk from pole 1 to pole 3
```

```
   move disk from pole 1 to pole 2
      HANOI : n=1 sp=3 tp=2 ap=1
      move disk from pole 3 to pole 2
move disk from pole 1 to pole 3
   HANOI : n=2 sp=2 tp=3 ap=1
      HANOI : n=1 sp=2 tp=1 ap=3
      move disk from pole 2 to pole 1
   move disk from pole 2 to pole 3
      HANOI : n=1 sp=1 tp=3 ap=2
      move disk from pole 1 to pole 3
```

בדוגמת הרצה זו הדפסנו את הפרמטרים האקטואליים בכל הפעלה של פונקצית hanoi(). כמו כן, הדפסנו את שורות הפלט בהסטה פנימה, ככל שעומק הרקורסיה גדל. בדרך זו ניתן לעקוב אחר סדר הקריאות לפונקציה הרקורסיבית ולבחון את הפלט בכל שלב.

ללא ביצוע הדפסות הביניים, מתקבל הפלט הבא:

```
HANOI TOWERS with    3 DISKS:

move disk from pole 1 to pole 3
move disk from pole 1 to pole 2
move disk from pole 3 to pole 2
move disk from pole 1 to pole 3
move disk from pole 2 to pole 1
move disk from pole 2 to pole 3
move disk from pole 1 to pole 3
```

תרגילים

1. מה מבצעת התוכנית הבאה?

```
        /*****     chp1509.c :קובץ תקליטור       *****/

#include "include\stdio.h"

char c ;

main()
{
   char *do_what (char *,int *) ;

      int i, array[25] ;
      char *ptr, c_array[]="AN EXAMPLE FOR RECURSION" ;

      for (i=0 ; i<25 ; ++i)
```

```
        array[i] = i ;

    ptr = do_what (c_array, array) ;
    printf("\nThe results are : %s and %d\n", ptr, c) ;
}

char *do_what (char *c_ptr, int *i_ptr)
  char *c_ptr ;
    int *i_ptr ;
{
  if (*c_ptr)
    *c_ptr = *(do_what(c_ptr+1, i_ptr+1)) ;
  else
    c = *c_ptr ;

  return(c_ptr - *i_ptr) ;
}
```

2. כתוב תוכנית רקורסיבית שקוראת תווים ומדפיסה אותם בסדר הפוך. קריאת התווים תתבצע עד שמגיעים לתו "." (נקודה). נסה לפתור זאת בדרך אחרת, המוצגת בבעיה 2 בפרק זה.

3. כתוב תוכנית רקורסיבית שקוראת תווים ומדפיסה אותם בסדר רגיל. תנאי עצירה כמו בשאלה 2.

4. כתוב פונקציה רקורסיבית לחישוב המכנה המשותף הגדול ביותר של שני מספרים שלמים (Greatest Common Denominator) באמצעות האלגוריתם של אוקלידס.

השיטה:

א. אם המספרים שווים - המכנה הגדול הוא הערך של המספרים.

ב. אם הם שונים - יש להפחית את המספר הקטן מהגדול ולהמשיך עם שני המספרים הבאים באותו אופן:

- ההפרש בין שני המספרים.
- המספר הקטן משני המספרים המקוריים.

5. הראה את סכימת הפרישה בקריאה לפונקציה ()FIB עבור N=3 ועבור N=4 (בדומה לדוגמה בפרק).

6. כתוב את הפונקציה ()strcmp המשווה בין שתי מחרוזות, כפונקציה רקורסיבית.

7. הראה את סכימת הפרישה בקריאה לפונקציה שבשאלה 6, עם הערכים "new-york" ו-"new":

```
strcmp ("new", "new-york");
```

8. כתוב פונקציה רקורסיבית לחישוב הנוסחה לקירוב של e^x. הפונקציה תקבל זוג מספרים ממשיים: eps ו-x, ותחשב את הסכום שלהם עד שהאיבר הנוסף יהיה קטן מ-eps.

$$e^x = 1 + X + \frac{x^2}{2!} + \frac{x^3}{3!} + \ ...$$

$$\frac{x^n}{n!} < eps$$

כדי לכתוב פונקציה רקורסיבית דרוש ערך נוסף, מהו?

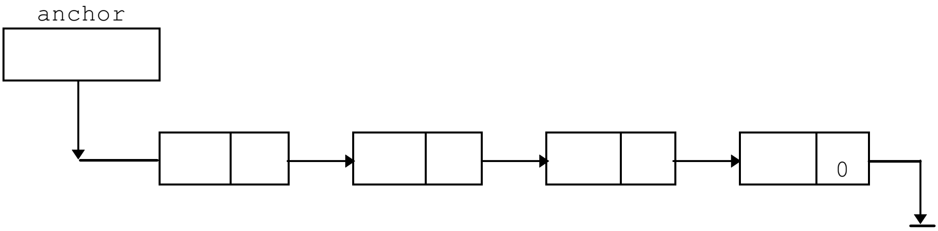
9. כתוב פונקציה רקורסיבית ()reach_one הנעזרת בעיקרון שלמדנו בשאלה קודמת: מכל מספר X ניתן לפרוש סדרה $X_1, X_2, \ldots X_n$ ומובטח ש-Xn יהיה 1. הסדרה היא:

$$X_1 = X$$

$$X_{i+1} = \begin{cases} X_i / 2 & X_i \text{ זוגי} \\ 3X_{i+1} & X_i \text{ אי-זוגי} \end{cases}$$

הפונקציה תדפיס את הסדרה המבוקשת, כאשר הערך 1 יודפס ראשון (בסדר הפוך).

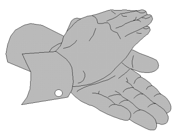
10. נתונה רשימה מקושרת המוצגת באופן סכימטי כך:



האיבר האחרון ברשימה מכיל את הערך 0 בשדה המצביע לאיבר הבא. ההגדרה הפורמלית של הרשימה:

```
struct l_list  {
         int  info;
         struct l_list *nxt_l;
               } *anchor;
```

כתוב פונקציה רקורסיבית ()reverse_list על פי כל השלבים שלמדנו. הפונקציה תהפוך את סדר קישור האיברים, כך שהאיבר האחרון יהיה הראשון (מוצבע על ידי anchor) והאיבר הראשון יהיה האחרון (והמצביע שלו יהיה 0).



משימות:

תרגילים 39-47.

פרק 16

קלט/פלט וקבצים

במערכת מחשב מחולק הזיכרון לשני סוגים עיקריים:

- זיכרון ראשי, או זיכרון מיידי.
- זיכרון משני.

הזיכרון הראשי - RAM (**R**andom **A**ccess **M**emory) קשור באופן ישיר למעבד המרכזי של המחשב. בזיכרון זה נטענת התוכנית ובו היא מצויה כל עוד היא מופעלת. המשתנים והמבנים המוגדרים בגוף התוכנית נמצאים גם הם בזיכרון זה. זמן הגישה לזיכרון הראשי, דהיינו זמן העברת נתון בין הזיכרון לבין המעבד - נמדד בננו-שניות (1 ננו-שנייה = 10^{-9} שנייה).

הזיכרון המשני מצוי בהתקן חיצוני. דוגמאות להתקנים כאלה הם **התקליטון** (דיסקט), **ודיסק קשיח**. הקשר בין המחשב לזיכרון זה אינו ישיר, אך המחשב יכול לפנות אליו בצורה דומה לזיכרון הראשי, ולאחסן או לשלוף מתוכו נתונים. זמן הגישה לזיכרון המשני נמדד במילי-שניות (1 מילי-שנייה = 10^{-3} שנייה). פרק זמן זה איטי פי מיליון מזמן הגישה לזיכרון הראשי.

לאור נתונים אלה, מדוע אנו זקוקים כלל לזיכרון משני? הסיבה נובעת מהתכונות השונות של הזיכרונות והמטרות להן הן יועדו. הזיכרון הראשי נועד להחזיק את התוכנית ואת הנתונים הדרושים לה במהלך העיבוד השוטף והוא מוגבל בנפחו. הזיכרון המשני נועד לאחסן את מאגרי הנתונים של המשתמש, הנמדדים בעשרות ובמאות מיליוני בתים. הזיכרון המשני זול מאוד באופן יחסי לזיכרון הראשי. שוני נוסף הוא, שהנתונים הנשמרים בזיכרון המשני אינם אובדים לאחר כיבוי המתח למחשב - לא כך לגבי הזיכרון הראשי.

כאשר יש כמות רבה של נתונים בזיכרון חיצוני וזקוקים לחלק מהם באופן תכוף במהלך ריצת תוכנית מסוימת, נשמור אותם בזיכרון הראשי מראש באופן קבוע, ככל שיתיר לנו המקום. ברוב המקרים קוראים לקטעי נתונים שונים לסירוגין במהלך העבודה, בהתאם לצורך.

ארגון הנתונים בזיכרון משני

כאשר אנו מציינים את המונח "זיכרון משני" נתייחס ברוב המקרים לתקליטון או לדיסק הקשיח. בסרט המגנטי נשתמש בעיקר לשמירת נתונים הסטוריים או לגיבוי ושמירת עותק של הנתונים שבדיסק הקשיח.

הנתונים אשר אוגרים בזיכרון המשני יכולים להיות **תוכניות מקור** (Source program) כפי שנכתבו על ידי המתכנת, תוכניות לאחר הידור וקישור ה**מוכנות להפעלה** (Object program), אוסף נתונים מספריים המייצגים מידע כלשהו כמו פרטים אישיים של עובדים במפעל, או ייצוג של תמונות גרפיות.

פריט המידע הבסיסי בעולם המחשב היא ה**סיבית** (bit), אשר יכולה לקבל את הערכים 1 או 0 בלבד. הסיביות מקובצות ב"חבילות" של שמונה הנקראות **בתים** (Bytes). בחשבון בינארי (בסיס 2) ניתן לייצג באמצעות בית אחד 256 ערכים שונים (0 עד 255). ניתן לייצג תווים במחשב באמצעות קוד ASCII. ראה בנספח טבלה המתארת את הסימנים והקוד המתאים לפי ASCII. על פי קודים אלה כל תו מיוצג על ידי בית אחד.

לכל טיפוס של משתנה יש ייצוג פנימי המורכב מבית אחד (טיפוס char), ועד שמונה בתים (טיפוס double). הנתונים בזיכרון נמצאים בצורת משתנים בודדים, או בצורת מבנים המכילים משתנים רבים ולכן גם מספר בתים רב יותר. הנתונים בזיכרון המשני נמצאים בקבצים בצורות ארגון שונות, כפי שנראה בהמשך. בדרך כלל, כל קובץ מכיל מידע בנושא מוגדר ובסופו יש ציון "סוף" על ידי תו הקרוי **E**nd **O**f **F**ile (או בקיצור EOF). לכל קובץ יש שם ייחודי וקיימת עבורו כניסה בטבלה מיוחדת הנקראת **ספריה** או **מדריך** (directory).

הקשר בין הקבצים לבין המשתמש מתבצע באמצעות **מערכת ההפעלה**. במערכת ההפעלה MS-DOS למשל, הנפוצה במחשבי IBM-PC, פקודת dir תגרום להקרנת הספריה של התקן הזיכרון המשני (תקליטון או דיסק). על המסך תתקבל רשימה של שמות הקבצים הקיימים באותו התקן אחסנה. כדי להציג את תוכן של קובץ מסוים עלינו לתת את הפקודה:

type <שם קובץ>**.**<סיומת>

חוקי הכתיבה של שמות הקבצים נקבעים על ידי מערכת MS-DOS. הצורה הכללית של שם קובץ הינה (הפעם כתבנו זאת באנגלית):

<filename>.<extension>

כאשר:

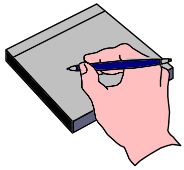
<filename> שם הקובץ, מורכב מ-1 עד 8 אותיות, תווים, מספרים או "_" (קו תחתון).

<extension> הסיומת מורכבת מ-1 עד 3 אותיות, מספרים או "_". הסיומת הינה רשות, כלומר אינה חייבת להופיע.

ניתן גם להצמיד משמאל לשם הקובץ את כינוי הכונן, אם דרוש: <כונן>:

כאשר <כונן> הינו A או B לציון דיסקט ו-C עבור דיסק קשיח. לדוגמה, הגדרת קובץ בכונן B:

```
B:myfile.txt
```



הערה!

*במערכת ההפעלה Windows 3.11 אפשר לקבל את רשימת הקבצים באמצעות **מנהל הקבצים**, וב-Windows95 - באמצעות **הסייר**.*

מערכות הפעלה כגון Windows95 או WindowsNT אינן מטילות מגבלות כה חמורות על שמות קבצים וניתן להשתמש במחרוזות ארוכות ובעלות מובן, כולל רווחים בין מילים.

לצורך תאימות עם כלי פיתוח שונים, מומלץ למרות זאת, להשתמש בכללים הנ"ל עד לפרישה רחבה של מערכות הפעלה אלו.

את קובצי הנתונים ניתן לארגן בצורות שונות. הפשוטה שבהן היא רצף של תווים מתחילת הקובץ ועד סופו. קובץ כזה נקרא **קובץ תמליל** או **קובץ טקסט** (Text File) ולמעשה, כל תוכנית מחשב שאנו כותבים (תוכנית מקור) היא קובץ מסוג זה. חלק מהתווים של קובץ התמליל הם תווים מיוחדים המסמנים סוף שורה, ותפקידם לחלק את הקובץ לשורות. התווים הללו הינם **CR** (ASCII-13) ו-**LF** (ASCII-10). כאשר מדפיסים אותם, או מציגים על המסך, הם גורמים לתווים העוקבים להתחיל בשורה חדשה.

Carriage Return - CR הוראת דילוג לתחילת השורה הנוכחית.

Line Feed - LF הוראת דילוג לשורה הבאה.

משתמשים בתווים הללו כתווים מיוחדים לסימון סוף שורה בקובץ, כדי שניתן יהיה להדפיס את הקובץ כמות שהוא, מסודר לפי שורות. לדוגמה, הקובץ examp1.txt מכיל את השורות הבאות:

```
This file contains
3 lines:
This is the third line.
```

									10										20								
T	h	i	s		f	i	l	e		c	o	n	t	a	i	n	s	C R	L F	3		l	i	n	e	s	:

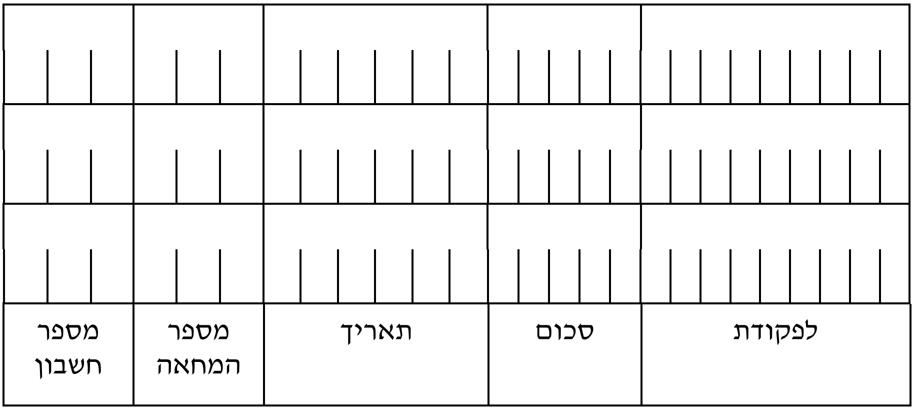
	30										40										50					
C R	L F	T	h	i	s		i	s		T	h	e		t	h	i	r	d		l	i	n	e		C R	L F

***תרשים 16.1:** טקסט עם תווי בקרת הדפסה*

מבנה הקובץ באחסנה יהיה כפי שמתואר בתרשים 16.1. הקובץ מורכב בדוגמה זו מרצף של 54 בתים. כל בית מייצג (בקוד ASCII) תו מסוים, כולל התווים המיוחדים לסימון סוף שורה וסוף קובץ.

שיטה נוספת לארגון נתונים בקובץ היא על ידי חלוקתו לרשומות. כל **רשומה** (Record) מורכבת מאוסף של שדות בדומה לתכולת **מבנה** (Structure) המצוי בזיכרון הראשי. גודל כל שדה תלוי בסוג הנתונים האגורים בו. הקובץ עצמו מכיל אוסף של רשומות הזהות אחת לחברתה במבנה ולמעשה, צורה זו של ארגון קובץ מקבילה למערך של מבנים הנהוג בזיכרון הראשי.

כפי שהוזכר, היתרון באחסנת מערך של מבנים באמצעות קובץ על גבי התקן משני, הוא באפשרות לאגור כמות גדולה מאוד של רשומות לזמן ארוך. ניקח לדוגמה קובץ המכיל תנועות בחשבון בנק. כל רשומה מכילה נתונים הקשורים להמחאה שהופקדה בחשבון ויש בה את השדות הבאים: מספר חשבון בנק (int), מספר המחאה (int), תאריך (char), סכום (float), לפקודת (char). מבנה הקובץ examp2.bnk מתואר בתרשים 16.2.



תרשים 16.2: *רשומות בקובץ*

שיטות גישה לקבצים

הגישה לנתונים המצויים בקבצים, שונה מגישה לנתונים בזיכרון הראשי של המחשב. ההבדלים נובעים מהעובדה שהזיכרון הראשי הינו מיידי, ולכן ניתן לגשת בצורה ישירה לכל תא זיכרון שבו. לקבצים המצויים בהתקן אחסנה חיצוני, יש לגשת באמצעות מנגנון ביניים מכני (בדיסקים - זרוע וראש קריאה/כתיבה), ולכן - זמן הגישה לנתונים ארוך יותר. מסקנה חשובה מדיון זה היא, שאין גישה מיידית לכל נתון בקובץ הנמצא בדיסק.

לכל קובץ שבו משתמשים במהלך התוכנית (לשם קריאה או כתיבה), קיים מצביע מיוחד המצביע אל הבית הבא בקובץ. כאשר נפתח קובץ לקריאה או לכתיבה יצביע המצביע על הבית הראשון שבו. כל קריאת בית מהקובץ תגרום לכך שהמצביע הזה יתקדם אוטומטית לבית הבא. באופן דומה, כאשר תוכנית המחשב יוצרת קובץ - יראה המצביע על המקום הבא בקובץ שבו ייכתב הבית הבא. לאחר הכתיבה, יתקדם המצביע באופן אוטומטי למקום הבא.

קיימות שתי שיטות גישה לקבצים: **גישה סדרתית** ו**גישה אקראית**.

הצורה הפשוטה ביותר לטפל בקובץ בקובץ היא ב**גישה סדרתית**. בשיטה זו מנצלים את המצביע האוטומטי לקובץ. תוכנית המחשב ערוכה לבצע את כל הפעולות הדרושות על הקובץ בסריקה סדרתית מתחילת הקובץ ועד סופו. שיטת גישה זו פועלת בדיסקים ובסרטים.

שיטת **גישה אקראית** אפשרית במתקני דיסקים בלבד. מכיון שלא תמיד ניתן לבצע את כל הפעולות הדרושות על קובץ בגישה סדרתית, קיימת אפשרות להזיז את המצביע ממקום למקום בקובץ לפי דרישת התוכנית, באופן אקראי. השליטה במיקומו של המצביע נעשית מתוך תוכנית המחשב, באמצעות פונקציה מיוחדת. לאחר מיקום המצביע על הבית הרצוי בקובץ, ניתן להמשיך בגישה הסדרתית (קריאה או כתיבה) עד להזזה הבאה.

מתכנני שפת C היו מעוניינים שהמהדר יהיה פשוט, קצר ויעיל. כתוצאה מכך **אין** בשפת C פקודות קלט/פלט. כדי לבצע פעולות אלו בקבצים, במסך, או במדפסת, צריך לקרוא לפונקציות ייעודיות, מוכנות מראש. לכל מהדר C קיימת ספריה של פונקציות (הספריה <stdio.h> במערכת Borland C, למשל), המכילה גם פונקציות לפעולות קלט/פלט בקבצים.

בשוק קיימים מהדרים רבים לשפת C וייתכנו הבדלים מסוימים בהגדרת הפונקציות בספריות השונות כפי שהוכנו על ידי היצרנים. נתאר להלן מספר פונקציות מייצגות, הנחשבות לסטנדרטיות. עליך לבדוק בספריית המהדר בו הינך משתמש מהי ההגדרה המדויקת לפונקציות הללו.

בספריית המערכת של C נחלקות הפונקציות המטפלות בקבצים לשתי קבוצות, אשר כל אחת מהן מטפלת בקבצים ברמה שונה:

א. פונקציות המספקות שירותי מאגר אוטומטי נקראות **Buffered I/O**.

ב. פונקציות שאינן מספקות שירותי מאגר נקראות **Unbuffered I/O**.

הפנייה אל קובץ באותה תוכנית צריכה להיות עקבית, באותה רמת שירות. קריאה או כתיבה מקובץ שנפתח ברמה הראשונה מתאפשר רק באמצעות הפונקציות השייכות לרמה הראשונה. וכך אם הקובץ נפתח ברמה השנייה - הוא יופעל בפונקציות של רמה זו.

פונקציות קלט/פלט עם מאגר

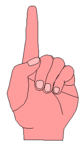
הפונקציות בקבוצה זו מטפלות בתחזוקת **מאגר הזיכרון** (Buffer, או **חוצץ**), אשר דרכו מתבצעות פעולות הקלט/פלט הקרויות **Bufferd I/O**. המצביע לקובץ מנוהל גם הוא על ידי פונקציות אלו.

שיטת העבודה היא:

א. כאשר תוכנית המחשב קוראת לפונקציה שתפקידה לשלוף את הבית הראשון מהקובץ, מתמלא המאגר בקטע שלם מהקובץ החל מהבית המבוקש.

ב. קריאה לפונקציה שנית, לשם הבאת בית נוסף, תגרום לשליפתו מהמאגר. לעובדה זו חשיבות רבה, שכן המאגר נמצא בזיכרון הראשי והגישה אליו הינה מיידית לעומת הגישה לדיסק למשל.

ג. לאחר קריאת הבית האחרון המצוי במאגר, הוא יתמלא בקטע הבא של הקובץ באופן אוטומטי. שיטה זו מאפשרת לקרוא את הקובץ באופן סדרתי, בית אחר בית במינימום גישות לדיסק, כי רוב העבודה מתבצעת דרך המאגר. כאשר כותבים לקובץ, תתבצע הכתיבה אל המאגר תחילה, ורק כשזה יתמלא, הוא ייכתב בבת-אחת לקובץ שבדיסק. מנגנון המאגר הזה "שקוף" לחלוטין למתכנת ומבוצע כולו באחריות הפונקציות.



שים לב!

יש אפשרות להכריח את המערכת לכתוב את תוכן המאגר לקובץ בדיסק על ידי הפקודה הבאה:

```
FILE *fP;
flush (fP);
```

פתיחה וסגירה של קבצים

בכדי שניתן יהיה להתייחס לקובץ מתוך תוכנית מחשב, יש צורך להגדיר את ההתקשרות אליו בנוהל **פתיחת קובץ** (Open). לאחר פתיחת הקובץ מתאפשרת פנייה של התוכנית אליו ומוקצים לו מאגר ומצביע. פונקצית הספריה המבצעת פתיחה של קובץ נקראת **fopen()**. המצביע לקובץ הינו הערך המוחזר על ידי הפונקציה fopen(). באמצעותו מתבצעות שאר הפעולות על הקובץ. תיאור הפונקציה:

```
FILE *fopen (fname, mode)
   char *fname, *mode;
```

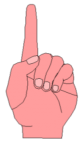
<fname> מחרוזת המכילה את שם הקובץ, כפי שמופיע בספריה (Directory).

<mode> מחרוזת המכילה אחד מן התווים "r", "w" או "a", אשר מגדירים את צורת העיבוד הרצויה:

"r" פתיחת קובץ לקריאה בלבד.

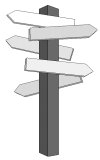
"w" פתיחת קובץ לכתיבה בלבד. אם הקובץ קיים הוא נמחק; אם הוא לא קיים הוא נבנה.

"a" פתיחת קובץ כדי להוסיף בסופו. אם הקובץ לא קיים הוא נבנה.



שים לב!

*בפתיחת קובץ טקסט מומלץ להוסיף את האות "**t**" (text) בבחירת mode, למרות שזו ברירת המחדל כלומר, לכתוב: "rt", "wt", "at".*



נקודת ציון!

ניתן לפתוח קובץ text לעדכון. המחרוזת של mode היא: "r+t".

אפשרויות נוספות להגדרת צורת עיבוד קובץ, בפרמטר <mode>:

r+ פתח קובץ קיים לקריאה/כתיבה.

w+ צור קובץ חדש לעדכון.

a+ פתח קובץ להוספה או צור קובץ חדש, אם הקובץ לא נמצא (הוספה - תוספת תווים לסוף הקובץ).

(r+b) rb פתח קובץ בינארי לקריאה.

(w+b) wb פתח קובץ בינארי לכתיבה.

(a+b) ab פתח קובץ בינארי להוספה.

כל תוכנית המכילה התייחסות לקבצים חייבת להכיל בתחילתה את ההוראה הזו:

```
#include <stdio.h>
```

<stdio.h> קובץ הגדרות בסיסי - **קובץ כותר** - שנועד לשימוש פונקציות הקלט/פלט של ספריית המערכת. בקובץ זה מוגדר הטיפוס מסוג **FILE**, תווים מיוחדים לצורך עיבוד הקובץ (כגון EOF המסמן סוף קובץ) ועוד.

דוגמה:

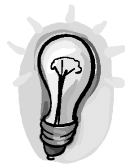
כדי לפתוח את הקובץ myfile.txt לקריאה נכתוב את משפטי התוכנית הבאים:

```
(1)   # include <stdio.h>
(2)   void main(void)
(3)   {
(4)   FILE *filep;
(5)   filep = fopen ("myfile.txt", "rt");
(6)   }
```

הסבר:

א. בשורה 4 המשתנה filep מוגדר כמצביע לקובץ, כדי שנוכל לשמור בו את תוצאות הפונקציה ()fopen. באמצעות filep תבוצענה הפניות לקובץ לאחר פתיחתו.

ב. בשורה 5 השתמשנו בפרמטר "rt", אך אפשר לכתוב "r" בלבד, כי טקסט הוא ברירת המחדל.



טיפ!

*מספר הקבצים שניתן לפתוח במקביל באותה תוכנית מחשב, הינו מוגבל. סיבה זו נובעת מכמות הזיכרון, אשר דרוש לכל קובץ עבור המאגר לאחר פתיחתו. בספריית המערכת נמצאת פונקציה המאפשרת לתוכנית ל**סגור קובץ** (Close) לאחר שסיימה להשתמש בו ולשחרר את הזיכרון הראשי שהוקצה לו.*

הפונקציה **לסגירת קובץ** מוגדרת כך:

```
FILE *fp;
fclose (fp);
```

בתהליך הסגירה של קובץ פלט, נכתב המאגר לקובץ, גם אם אינו מלא. פעולה זו מעדכנת את הקובץ, לפני ניתוק הקשר בינו לבין התוכנית.

כאשר תוכנית מחשב מסתיימת באורח מסודר, כלומר ללא תקלה, נסגרים באופן אוטומטי כל הקבצים שנשארו פתוחים.

נציג דוגמה פשוטה לפתיחת קובץ על ידי פנייה אליו, בדיקה אם הוא פתוח, סגירתו ויציאה מהתוכנית. היציאה מהתוכנית הינה עם קוד 0 אם היא בוצעה כראוי. אם היתה שגיאה בפתיחה - יוצאים עם קוד 1.

```
#include <stdio.h>

int main(void)
  {
     File *in;
          /* הבדיקה המוודאת כי פתיחת הקובץ תקינה חשובה ביותר ובלתי נמנעת */
     if ((in = fopen("myfile.txt, "rt")) == NULL
          {
           printf("Cannot open input file.\n");
           return (1);
          }
     fclose(in);
     return(0)
}
```

שים לב!

יש לבדוק את פתיחת הקובץ, מכיון שפתיחת קובץ לא מוצלחת עלולה לגרום לתקלות חמורות, כמריחת זיכרון במקרה של כתיבה לקובץ. אסור לטפל בקובץ שלא נפתח כראוי.

קריאה וכתיבה של תווים בודדים

צמד הפונקציות הבסיסי לקריאה וכתיבה של תווים (בתים) בקובץ הן **fgetc()** ו-**fputc()** בהתאמה.

הפונקציה fgetc() לקריאה מוגדרת כך:

```
char fgetc (fp)
FILE *fp ;
```

תפקיד הפונקציה להביא לתוכנית את התו הבא בקובץ המוצבע על ידי fp. קידומו של המצביע יבוצע באחריות fgetc(), לאחר פעולת הקריאה.

הפונקציה fputc() לכתיבה מוגדרת כך:

```
fputc (c, fp)
char c;
FILE *fp;
```

תפקיד הפונקציה לכתוב את התו המוכל ב-c אל הקובץ המוצבע על ידי fp. קידומו של המצביע יבוצע לאחר הכתיבה.

באמצעות ארבע הפונקציות fopen(), fgetc(), fputc() ו-fclose() ניתן לכתוב תוכנית מחשב קצרה המבצעת העתקה מקובץ לקובץ, בדומה לפקודה copy של MS-DOS. הפונקציות fopen ו-fclose אינן הכרחיות.

שם התוכנית יהיה ccopy ונכתוב אותה בקובץ ששמו chp1601.c:

```
        /*****      chp1601.c :קובץ תקליטור       *****/
#include <stdio.h>
void main(void)
{
  FILE *fromfile, *tofile;
  char c ;
  char ffname[20], tfname[20];
                                              /* קליטת שמות קבצים */
  printf("Enter file name to copy-");
  scanf("%s", ffname);
  printf("\nEnter new file name -");
  scanf("%s", tfname);
                                                 /* פתיחת קבצים */
  fromfile = fopen(ffname, "r");
  tofile = fopen(tfname, "w");
```

```
/* העתקה */
  while ((c=fgetc(fromfile)) != EOF)
    fputc(c, tofile);

/* סגירת קבצים */
  fclose (fromfile) ;
  fclose (tofile) ;
}
```

הסבר:

בתוכנית זו התעלמנו מהמקרה בו הקובץ ffname אינו קיים כלל. אם הוא אינו קיים, תחזיר הפונקציה ()fopen את הערך 0 (או NULL, כפי שהוא מוגדר בקובץ stdio.h). כדי לבדוק אם קובץ הקלט (from) וקובץ הפלט (to) קיימים, צריך לכתוב את הפקודות האלו, אשר מחזירות קוד שגיאה וגם מודיעות על כך.

```
if (fromfile == NULL) printf("Error"); return(1);
if (tofile == NULL) printf("Error"); return(1);
```

תרגיל

השתמש בדוגמה זו לשיפור התוכנית chp1601.c, כך שתבדוק את מצב פתיחת הקבצים, ובמקרה של שגיאה תדווח הודעה מתאימה ותסיים ללא ביצוע העתקה. נסה לשפר עוד את התוכנית ולאפשר למשתמש להחליף קובץ פלט ולחזור על הפעולה.

קובצי תמליל

קובצי תמליל (Text files) מכילים שורות של תווים אשר מסתיימות בצמד התווים CR/LF. בספריית המערכת קיים צמד פונקציות, שנועד להקל על הטיפול בקבצים אלה. תפקיד הפונקציות ()fgets ו-()fputs הוא לקרוא ולכתוב שורה שלמה בהתאמה (שים לב, שמות הפונקציות מסיימים ב-s, ולא ב-c כמו קודם).

פונקציית הקריאה ()fgets

```
char *fgets (char *line, int Max, FILE *fp)
```

הפונקציה קוראת את השורה הבאה בקובץ המוצבע על ידי fp ומכניסה אותה למחרוזת המוצבעת על ידי line. המחרוזת כוללת גם את התווים CR/LF. המשתנה Max מגדיר את המספר המקסימלי של תווים שניתן לקרוא. תפקידו לדאוג שלא תהיה גלישה מאורך נתון, במידה וצמד התווים CR/LF אינו נמצא בקובץ. הפונקציה תחדל מקריאה כאשר תיתקל בצמד התווים CR/LF, או כאשר תקרא Max תווים.

המחרוזת שמוחזרת לתוכנית הקוראת מכילה NULL מסיים, שנוסף אוטומטית על ידי הפונקציה ()fgets. הפונקציה ()fgets מחזירה מצביע לתחילת line. כאשר מגיעים לסוף הקובץ (EOF) מחזירה הפונקציה NULL.

דוגמה להפעלת הפונקציה ()fgets על הקובץ examp1.txt :

```
(1)    # include <stdio.h>
(2)    void main(void)
(3)    {
(4)      FILE *filep;
(5)      char m[100];

(6)      filep = fopen ("examp1.txt", "r");
                 if (filep == Null)
                  {
                     printf("Cannot open input file.\n");
                     return(1);
                  }
(7)      fgets (m, 100, filep);
(8)    }
```

למחרוזת m ייקלטו הנתונים הבאים :

m:	C	a	n	n	o	t		o	p	e	n		i	n	p	u	t		f	i	l	e			CR	LF		

פונקציית הכתיבה ()fputs:

fputs (char *line, FILE *fp);

הפונקציה כותבת את המחרוזת המוצבעת על ידי line, אל הקובץ המוצבע על ידי fp. היא מוסיפה את התווים CR/LF בסוף כל שורה. התו NULL הנמצא בסוף השורה אינו נכתב לקובץ.

נציג את הפונקציה ()fgets בשפת C :

```
        /*****      chp1602.c  :קובץ תקליטור       *****/

char *fgets(char *line, int max, FILE *fp)
{
  char c, *s ;

  s = line ;
  while (max-- && ((c=fgetc(fp)) !=EOF))
    if ((*line++ = c) =='\n')
       break ;
    *line = NULL ;
    return((c==EOF && line==s) ? NULL : s);
}
```

הסבר:

א. בלולאת while ממלאים את המחרוזת line בתווים הנקראים מהקובץ באמצעות הפונקציה fgetc().

ב. תנאי העצירה של הלולאה הם: זיהוי מצב של סוף קובץ (EOF), או קריאת max תווים.

ג. אם נקלט תו סוף שורה ('\n') יוצאים מהלולאה על ידי משפט break.

ד. לאחר סיום הלולאה משרשרים תו NULL לסוף המחרוזת.

ה. הערך המוחזר הוא מצביע לתחילת המחרוזת, או NULL, אם התו הראשון היה EOF.

תרגיל

כתוב באופן דומה את הפונקציה fputs(), והיעזר בפונקציה fputc() (כפי שבדוגמת הקריאה נעזרנו בפונקציה fgetc()).

מסוף המחשב כקובץ סטנדרטי

פונקציות הקלט/פלט של C מתייחסות למסוף המחשב כקובץ. בכל הרצת תוכנית נפתחים באופן אוטומטי שלושה קבצים מיוחדים. הפנייה אליהם נעשית באמצעות המצביעים **stdin** לקלט, **stdout** לפלט, **stderr** לפלט שגיאות. קבצים אלה מוגדרים בקובץ stdio.h.

באמצעות stdin ניתן לקרוא תווים המוקשים במקלדת, ובאמצעות stdout ו-stderr ניתן לכתוב תווים למסך. פונקציות הקלט/פלט שלמדנו קודם, ישימות גם עבור שלושת הקבצים האלה: ניתן להשתמש בפונקציות fgetc(), fgets(), fputc(), ו-fputs().

בנוסף לפונקציות printf() ו-scanf() קיימות בשפת C הפונקציות getch() ו-putch() המשמשות לקליטה וכתיבה של תווים למסך. למעשה קיימת ההקבלה הבאה:

```
getch() - fgetc(stdin)
putch(c) - fputc(c,stdout)
```

נכתוב לדוגמה, תוכנית ctype.c עבור פקודת DOS להצגת תוכן קובץ על המסך:

```
type <שם קובץ>
```

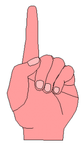
```
        /*****      chp1603.c :קובץ תקליטור       *****/
#include <stdio.h>
void main(void)
{
  FILE *fp;
  char c;
  char fname[20];
  printf("Enter file name to type--");
```

```
  scanf("%s", fname);

  fp = fopen(fname, "r");

  while ((c=fgetc(fp)) != EOF)
    fputc(c, stdout);

  fclose (fp) ;
}
```



שים לב!

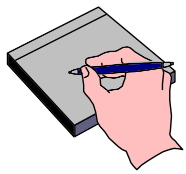
אין צורך לפתוח את הקבצים stdin ו-stdout, כי הם נפתחים אוטומטית בזמן הרצת כל תוכנית.

תרגיל

כתוב את התוכנית makefile.c, שתפקידה לקלוט שם של קובץ ולהכניס לתוכו נתונים באמצעות המקלדת. כיצד תסיים את התוכנית?

תרגיל

כתוב נוסח נוסף לתוכנית chp1603.c (המקבילה לפקודה type של DOS), אשר יתאים לקובץ מסוג text. השתמש בפונקציות ()fputs ו-()fgets.



הערה!

בוודאי שמת לב לעובדה, שיש שני מצביעים שונים לביצוע פלט למסוף: stdout ו-stderr. הסיבה לכך תוסבר בהמשך, אך עתה נאמר, ש-stdout משמש לביצוע פלט תקין של תוכנית ו-stderr משמש לפלט של הודעות שגיאה.

קלט/פלט עם פורמט

הפונקציות ()scanf ו-()printf מבצעות קלט ופלט למסוף. קיימות שתי פונקציות משלימות לעבודה עם קבצים, ()fscanf ו-()fprintf, אשר זהות ל-()scanf ו-()printf, פרט לעובדה שהפרמטר הראשון שלהן הוא מצביע לקובץ שהן פועלות עליו. לאור זאת, אפשר לכתוב כך את המבנה הפורמלי של הפקודות האלו:

```
printf (<מחרוזת בקרה>, <משתנה>, <משתנה>);
fprintf (stdin (<מחרוזת בקרה>, <משתנה>, <משתנה>);
```

הפונקציות ()scanf ו-()fscanf מוגדרות באופן דומה.

התוכנית הבאה תדגים את השימוש ב-()fscanf. התוכנית מדפיסה את שדות קובץ הנתונים examp2.bnk:

```
        /*****     chp1604.c  :קובץ תקליטור      *****/
#include <stdio.h>
void main(void)
{
   FILE *fp;
   int account, chknum;
   float sum;
   char chkdt[8], name[20];
   fp = fopen("examp2.bnk", "r");
   printf
      ("Account Check   Date   Sum   In-order-of\n");
   while (fscanf (fp, "%d, %d %s %f %s ",&account,&chknum,
                            chkdt, &sum, name) != EOF)
     printf("%5d   %3d %s %7.2f %s\n", account,
                            chknum, chkdt, sum, name);
   fclose (fp) ;
}
```

גישה אקראית לקבצים עם מאגר

הפונקציה ()fseek, מאפשרת להציב את המצביע לקובץ, כך שיצביע לכל בית רצוי. לאחר מיקום המצביע ניתן להמשיך בגישה סדרתית מהמקום בקובץ שבו הוא נמצא. כאשר המצביע מתקדם בקובץ ומגיע לנתונים שאינם נמצאים במאגר, הפונקציה מבצעת באופן אוטומטי קריאה של נתונים חדשים מתוך המשך הקובץ שלפניה.

הפונקציה מוגדרת כך:

fseek (FILE *fp, long offset, int origin);

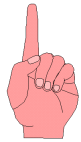
fp הוא מצביע לקובץ.

offset כמות הבתים שיש לדלג (היסט חיובי או שלילי).

origin מקבל אחד משלושה ערכים 0, 1, 2:
"0" ההיסט יימדד מתחילת הקובץ.
"1" ההיסט יימדד מהמיקום הנוכחי של fp.
"2" ההיסט יימדד מסוף הקובץ.

דוגמאות לשימוש בפונקציה:

- דלג לתחילת הקובץ `fseek (fp, 0L, 0)`
- דלג לסוף הקובץ `fseek (fp, 0L, 2)`
- דלג קדימה 100 בתים `fseek (fp, 100L, 1)`
- דלג לבית ה-4 בקובץ `fseek (fp, 4L, 0)`



שים לב!

*ערך **ההיסט** (Offset) הינו מטיפוס long. כאשר משתמשים בקבוע, יש לציין את הטיפוס על ידי הצמדת L לסופו, כדי שהמהדר יתרגם אותו כראוי לקבוע מסוג long.*

פונקציות קלט/פלט ללא מאגר

פונקציות קלט/פלט אשר אינן משתמשות במאגר, מספקות רמת שירות נוחה פחות למתכנת, אך מאפשרות לו לקבוע ביתר גמישות את אופן הפנייה לקבצים. מכיון שפונקציות אלו אינן פועלות באמצעות מאגר סטנדרטי אוטומטי, הן משתמשות בשיטת הגישה הישירה בלבד. על המתכנת לספק לפונקציה מאגר שיוגדר בתוכנית הקוראת, כדי שאפשר יהיה לקלוט לתוכו את הנתונים מתוך הקובץ בדיסק, או להכין בו את הנתונים לשם כתיבה לקובץ. מאגר זה יכול להיות מערך, מבנה או מחרוזת. פרט לכך, העבודה עם פונקציות אלו דומה לעבודה שהוצגה בסעיף הקודם:

- פתיחה של הקובץ לפני שמשתמשים בו.
- אפשרות להקפיץ את המצביע לכל מקום בקובץ, וממקום זה להתקדם בגישה סדרתית.
- הפניות לקובץ אינן נעשות באמצעות מצביע (FILE*), אלא על ידי מספר ערוץ המוגדר כמספר שלם.

הערוץ (channel) הוא מבנה נתונים המכיל נתונים על הקובץ ובכללם המיקום של המצביע בקובץ.

פתיחה וסגירה של קבצים

הפונקציה ()open מיועדת לפתיחת קובץ:

```
int  open (char *fname, int rwmode)
```

rwmode	**0**	קריאה
	1	כתיבה
	3	שילוב קריאה + כתיבה

הפונקציה ()open מחזירה ערך שלם, שהוא מספר הערוץ שנפתח עבור הקובץ. הפניות לקובץ יבוצעו דרך הערוץ הזה. היא מחזירה ערך 1- כאשר יש תקלה במהלך הפתיחה של הקובץ, כמו למשל כשהקובץ אינו קיים, או כשיש תקלה בדיסק.

כדי לפתוח קובץ חדש לכתיבה, יש ליצור אותו. **פונקציית הספריה ()fopen** מספקת שירות זה באופן אוטומטי, כשמנסים לפתוח קובץ שאינו קיים. הפונקציה ()open מתייחסת לקובץ קיים או לקובץ שנוצר תחילה על ידי ה**פונקציה ()creat**:

```
int  creat (char *fname) ;
```

<fname> הוא שם הקובץ שברצוננו ליצור. הפונקציה מחזירה את מספר הערוץ שנפתח לקובץ לצורך כתיבה. הערך 1- מסמל תקלה כלשהי.

שים לב!

הפעלת creat() על קובץ קיים תגרום למחיקתו ופתיחת קובץ חדש, בעל שם זהה, לכתיבה.

הפונקציה close() מיועדת לסגירת קובץ:

```
int  close (int fn) ;
```

פונקציות לקריאה וכתיבה

הפונקציות הבסיסיות לקריאה ולכתיבה הן read() ו-write() והגדרתן היא:

```
int write (int fn, char *s, unsigned b);
int read (int fn, char *s, usigned b);
```

<fn> מספר הערוץ לקובץ.

<b> מספר הבתים המבוקש לקריאה/כתיבה.

<s> מצביע למאגר המסופק על ידי התוכנית.

הערך המוחזר הוא מספר מטיפוס int, המציין את מספר התווים (בתים) שהועברו מהקובץ למאגר שבתוכנית, או מהמאגר אל הקובץ.

הפונקציה read() קוראת b בתים מהקובץ אל המאגר s. היא אינה מוסיפה NULL בסוף המאגר, מכיון שקיימים יישומים רבים בהם המאגר אינו מוגדר כמחרוזת תווים, אלא כמבנה או כמערך. מספר הבתים בפועל (n) יהיה שווה ל-b, פרט למצב של EOF לפני תום הקריאה, שבו יתקיים n < b. ערך מוחזר 1- מסמל תקלה בקריאת הקובץ, אשר יכולה להיגרם מתקלה מכנית או אלקטרונית, או כתוצאה מפנייה לערוץ שאינו קשור עם קובץ כלשהו.

פעולת **הפונקציה write()** הפוכה: כותבים מן המאגר אל הקובץ. אם לא היתה תקלה בפעולת הפונקציה, היא תחזיר תמיד את הערך b, המציין את מספר הבתים שנכתבו. הגודל של הערך b וגודל המאגר נתונים לשיקול דעתו של המתכנת, ואינם מוגבלים.

אזהרה!

אין לקרוא לפונקציה read() עם מספר b שערכו גדול מגודל המאגר s, כי דבר זה עלול לגרום לפגיעה בזיכרון שמעבר למאגר. הפונקציה אינה כותבת NULL בסוף המאגר, ולכן יש לשים לב לעובדה זו, אם רוצים להתייחס ל-s כמחרוזת.

צמד הפקודות הבא מטפל בבעיה זו:

```
n = read (fn, s, b);
s[n] = NULL;
```

בהפעלת הפונקציה ()write מומלץ גם כן, שערכו של b יהיה קטן או שווה לגודל המאגר, כדי שלא לכתוב לקובץ תווים לא רצויים.

כדי לקרוא תו בודד מהקובץ לתוך משתנה מסוג char, יש לפעול כך :

```
char c;
n = read (fn, &c, 1);
```

דוגמה:

נכתוב תוכנית Fcopy.c (Fast - copy) הדומה ל-ccopy.c, שתעתיק מקובץ אחד לאחר במהירות גבוהה יותר, באמצעות שימוש בפונקציות ללא מאגר. נגדיר עבור הפונקציות מאגר בגודל מקסימלי.

פתרון:

```
        /*****     chp1605.c :קובץ תקליטור      *****/
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
void main(void)
{
  int n, fromfile, tofile;
  char fbuf[32000];
  char ffname[20], tfname[20];
  printf("Enter file name to copy-");
  scanf("%s", ffname);
  printf("\nEnter new file name -");
  scanf("%s", tfname);
  if ((fromfile = open(ffname,0)) == -1)
    {
    printf("Can't find source file %s\n", ffname);
    exit(0);
    }
  if ((tofile = creat(tfname,0xffff)) == -1)
    {
    printf("Error trying to create target %s\n", tfname);
    exit(0);
    }
  while ((n=read(fromfile,fbuf,32000)) >0)
    write(tofile, fbuf, n);
  if (n==0)
    printf("Done copying \n");
  else
    printf("Error while copying\n");
  close (fromfile) ;
  close (tofile) ;
}
```

הסבר:

exit() הינה פונקציית ספריה מיוחדת, שתפקידה לסיים תוכנית בצורה מסודרת: לסגור את הקבצים הפתוחים ולהודיע על סיום, או על מצב שגיאה, בהתאם לצורך. המהדר קורא לפונקציה זו לאחר סיום הפונקציה הראשית main().

גישה אקראית לקובץ ללא מאגר

הפונקציה lseek() הפועלת על קובץ פתוח ללא מאגר, מאפשרת להציב את המצביע לקובץ על בית מסוים.

```
long lseek (int fn, long offset, int origin);
```

<fn> מספר הערוץ לקובץ.

<offset> מספר הבתים לדילוג.

<origin> הנקודה ממנה נמדד ההיסט:
- **"0"** מתחילת הקובץ.
- **"1"** מהנקודה הנוכחית.
- **"2"** מסוף הקובץ.

שים לב!

lseek() מוגדרת בצורה זהה ל-fseek(), למעט העובדה שהערך הראשון שלה הוא מספר ערוץ לקובץ. הפעולה של lseek() מהירה יותר לעומת fseek(), כיון שאינה מחדשת תוכן מאגר בעקבות פעולת הדילוג.

קבצים ללא מאגר ומסוף המחשב

בעת עיבוד של קובץ כלשהו, אין לערב בין פונקציות קלט/פלט הפועלות עם מאגר לבין הפונקציות הפועלות ללא מאגר. למגבלה זו יש יוצא דופן אחד והוא **מסוף המחשב** (consol). הזכרנו קודם, שהמסוף הוא קובץ לכל דבר, וכל הפעלה של תוכנית C מלווה בפתיחת שלושה קבצים המוצבעים על ידי stdin, stdout ו-stderr.

במקביל למצביעים אלה ניתן לפנות לקבצים באמצעות הערוצים 0, 1, ו-2 בהתאמה. לקבצים שיפתחו באופן מפורש על ידי התוכנית יוקצו הערוצים ממספר 3 ומעלה. בעת השימוש במסוף אפשר להשתמש בשתי שיטות הגישה, ואך ורק במקרה זה.

נכתוב תוכנית בשם echo.c (הד, Echo), אשר תקרין על המסך כל תו שנקיש במקלדת, עד לקבלת התו ^C (ASCII 03). התו ^C מתקבל על ידי החזקת מקש Ctrl לחוץ - והקשת התו C.

```
#include <stdio.h>
#include <io.h>

void main(void)
{
   char c ;

   while ((c=fgetc(stdin)) != 03)
      fputc (c, stdout) ;
}
```

תוכנית זו מקבילה לתוכנית הבאה:

```
        /*****      chp1606.c :קובץ תקליטור       *****/

#include <stdio.h>
#include <io.h>

void main(void)
{
   char c ;

   while (read(0,&c,1), c != 3)
      write(1, &c, 1);
}
```

הסבר:

א. שים לב שתנאי לולאת while מכיל ביטוי הכולל פסיק, ולכן תתבצע תחילה קריאת המשתנה c באמצעות הפונקציה ()read.

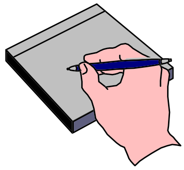
ב. השוואת המשתנה c ל-03 קובעת את המשך ביצוע הלולאה.

ניתוב מחודש של קלט/פלט סטנדרטי (redirection)

הערוצים 0, 1 ו-2 והמצביעים stdin, stdout ו-stderr מאפשרים להתייחס למסוף כקובץ, כדי לקרוא ממנו תווים (מקלדת) ולכתוב עליו תווים (מסך). מערכות ההפעלה התומכות בשפת C, כדוגמת UNIX או MS-DOS, מאפשרות לנתב ערוצים אלה בזמן הפעלת התוכנית לקבצים ממשיים הנמצאים בדיסק.

כאשר מפעילים את התוכנית, על ידי הקלדת שמה, היא גורמת לכך שהערוצים הסטנדרטיים לקלט ופלט יקושרו למקלדת ולמסך בהתאמה.

אם נכתוב בשורת ההפעלה infile > יקושר ערוץ 0 (מצביע stdin) לקובץ infile, במקום למקלדת. בצורה דומה ניתן לפתוח קובץ פלט בדיסק על ידי כתיבת outfile >.



הערה!

אין צורך להדר את התוכנית פעם נוספת, כי מערכת ההפעלה מכוונת כראוי את הערוצים באופן חיצוני.

ערוץ השגיאות stderr מיועד לניתוב של פלט הודעות השגיאה אל המסך. אם נבצע ניתוב מחודש לקובץ הפלט stdout, נמשיך עדיין לקבל הודעות אלו על המסך. רק קריאה מפורשת באמצעות הפקודה outfile*> תנתב את שני ערוצי הפלט לקובץ outfile. אין דרך לנתב את stdout ו-stderr לשני קבצים שונים.

דוגמה לשימוש בניתוב מחדש, ניתן להציג באמצעות echo.c. הפעלת התוכנית מתבצעת על ידי הקשת echo. כל תו שנקיש יופיע על המסך עד הקשת C^ כעת ננתב מחדש את הפלט:

```
echo > efile.txt
```

כעת, כל תו שנקיש ייכתב לקובץ efile.txt.

ניתוב מחדש של הקלט מאפשר להציג פתרון נוסף של התוכנית type המשמשת להדפסת תוכן קובץ:

```
echo < efile.txt
```

הקלט ינותב מהקובץ ויוצג על המסך. יש להכין מראש את התו C^ בסוף הקובץ efile.txt כדי שהתוכנית תסתיים.

ניתוב משולב של קלט/פלט מאפשר להשיג בעזרת echo, פתרון שונה לתוכנית copy להעתקת קובץ:

```
echo < Infile.txt > ofile.txt
```

הקלט מקובץ Infile.txt ייכתב לקובץ ofile.txt שמוצבע על ידי הערוץ stdout. גם כאן חשוב להכין את התו C^ בקובץ Infile.txt.

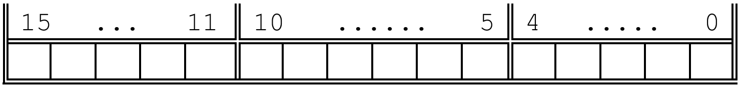
ניתוב קלט/פלט שימושי מאוד כאשר כותבים תוכניות עזר, והוא מאפשר לנצלן בגמישות מרבית.

מבנה קובץ DOS

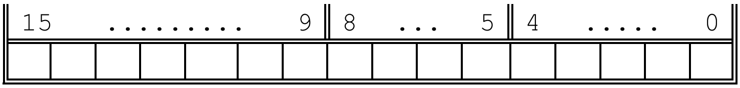
לאחר שלמדנו את נושא מבנים ועקרונות הטיפול בקבצים, נלמד על מבנה קובץ כפי שמערכת DOS מתייחסת אליו.

```
struct ffblk
      {
       char ff_reserved[21];          שמור עבור DOS
       char ff_attrib;                תכונות הקובץ
       int  ff_ftime;                 זמן יצירת הקובץ
       int  ff_fdate;                 תאריך יצירת הקובץ
       long ff_fsize;                 גודל הקובץ
       char ff_name[13];              שם הקובץ
      };
```

ff_ftime: 16 סיביות



ff_fdate: 16 סיביות



attr: תכונות קובץ

תיאור	קבוע
FA_RDONLY	קריאה בלבד
FA_HIDDEN	קובץ נסתר
FA_SYSTEM	קובץ מערכת
FA_LABEL	תווית
FA_DIREC	ספריה
FA_ARCH	ארכיב

מוחזר: בהצלחה ערך 0.
בכישלון ערך (-1).

בתוכנית הבאה נלמד כיצד ניתן להגיע למידע אודות קובץ, אשר נוצר במערכת DOS.

```
        /*****      fileread.c  :קובץ תקליטור       *****/

                                      /* DOS  -הצג מידע אודות קובץ הנוצר ב  */

#include <stdio.h>
#include <conio.h>
#include <dir.h>

/* ----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ---------------------------------------------------- */
int main(void)
 {
  struct ffblk fb;
  int find, dd, mm, yy, hh, mi, ss;

   clrscr();
  printf("autoexec.bat : הצג אינפורמציה על הקובץ\n\n");
                                              /* חפש את הקובץ על הדיסק  */
  find = findfirst("c:\\autoexec.bat", &fb, 0);
  if (!find)                                          /* אם מצאנו */
    {
                            /* חשב תאריך וזמן יצירת הקובץ על פי נוסחאות קבועות  */
      mm = (fb.ff_fdate >> 5) & 0x0F;          /* חשב חודש  */
      dd = fb.ff_fdate & 0x1F;                 /* חשב יום   */
      yy = ((fb.ff_fdate >> 9) & 0x7F) + 80;  /* חשב שנה    */
      hh = (fb.ff_ftime >> 11) & 0x1F;        /* חשב שעה    */
      mi = (fb.ff_ftime >> 5) & 0x3F;         /* חשב דקה    */
      ss = (fb.ff_ftime & 0x1F) * 2;           /* חשב שניה   */
        /* DOS -ב  dir  הדפס תוצאות במבנה דומה לפקודת  */
      printf("%-13s    %7ld %02d-%02d-%02d
              %02d:%02d:%02d\n", fb.ff_name, fb.ff_fsize,
              dd, mm, yy, hh, mi, ss);
    }
  printf("\nPress any key\n");
  getch();
  return(0);
 }
```

תוצאות ריצת התוכנית:

```
autoexec.bat : הצג אינפומציה על הקובץ

AUTOEXEC.BAT         858 09-12-96  22:25:04

Press any key
```

תרגילים

1. כתוב תוכנית שתבנה קובץ סטודנטים המכיל את הרשומות שלפניך:

שם	מספר זהות	מין	מספר שעות	ממוצע
moshe naor	5424040	M	120	95
ruth barzel	5323051	F	100	82
yonatan carmel	2350901	M	90	75
meir shani	3017975	M	100	93
maya ron	3357974	F	80	85

הגדרת הנתונים:

- מחרוזת של 30 תווים.
- מספר ת.ז.: שלם מטיפוס long.
- מין: תו בודד.
- מספר שעות: מספר שלם.
- ממוצע: מספר שלם.

2. כתוב תוכנית שתסרוק את הקובץ שבשאלה 1 ותדפיס את כל הרשומות שממוצע הציונים בהן גבוה מ-90.

3. כתוב תוכנית שתסרוק את הקובץ שבשאלה 1. כאשר התוכנית תגיע לרשומה המכילה מספר שעות גבוה או שווה ל-100, היא תעדכן בקובץ את מספר הזהות ל-0.

4. כתוב תוכנית שתבצע בעזרת פונקציות מתאימות את הפעולות הבאות:

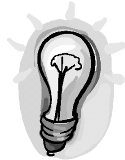
   א. העתקת קובץ לשני קבצים במקביל.

   ב. הוספת קובץ בהמשך של קובץ אחר.

   ג. השוואת קבצים והדפסת ההבדלים ביניהם.

   ד. חיפוש מחרוזת כלשהי בקובץ.

כתוב את התוכנית הזו פעמיים: בעזרת פונקציות עם מאגר ובעזרת פונקציות ללא מאגר. השווה בין זמני הביצוע של הפעולות השונות.



הנחיה!

עליך למדוד בעזרת שעון שניות או שעון עצר את זמני ההתחלה והסיום של התוכנית ולחשב את ההפרש בשני המקרים. שים לב שעליך לבחור למטרה זו קובץ גדול, אחרת לא תהיה משמעות למדידה שלך.

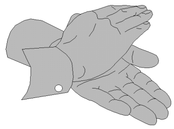
5. נתון קובץ המכיל מלל מוצפן לפי השיטה הבאה:

- ❖ הסימן לשינוי שיעור ההיסט הוא "$", ולידו מספר דו-ספרתי המציין את השיעור fact.
- ❖ **לדוגמה:** $02 מציין ששיעור ההיסט ממקום הציון ועד למקום הציון הבא הוא 2 (fact=2).
- ❖ התו הבא שייך למלל.
- ❖ מספר שלם דו-ספרתי n המציין את מכפלת ההיסט.
- ❖ תו המלל הבא נמצא במרחק fact*n מהתו האחרון.

דוגמה לקובץ המצפין את המילה "CODE":

```
$02C04DEW6IO02M$01D03E
```

כתוב תוכנית שמתרגמת קובץ המוצפן בשיטה זו, ומדפיסה את המלל המפוענח בקובץ אחר. השתמש בפונקציות ()fscanf ו-()fprintf כדי לפתור את התרגיל. את הקובץ המוצפן ניתן להכין באמצעות כל עורך-תמלילים.



משימות:

תרגילים 48-51.

פרק 17

פסיקות ועבודה עם BIOS

כפי שלמדנו על מבנה המחשב (ראה נספח) המשימה הראשונה שעליה צריכות תוכניות ROM לפקח, היא **הפעלת המחשב**. בעת ההפעלה, מפעילות הפונקציות שב-ROM תוכניות לבדיקת היחידות ההיקפיות, גורמות להפעלתן, טוענות ערכים ראשוניים בטבלת הפסיקות. בדיקת תקינות הציוד ההיקפי נקראת **POST** (**P**ower **O**n **S**elf **T**est), ולאחר מכן מגיע **תהליך האתחול** (Boot), שלאחריו עובר הפיקוח למערכת ההפעלה (DOS או Windows95).

ROM BIOS הוא חלק ב-ROM שנמצא בשימוש כל זמן שהמחשב פועל, ותפקידו לתת שירותים בסיסיים להפעלת המחשב ולבצע משימות שונות הקשורות ליחידות ההיקפיות. כלומר, BIOS נמצא בין החומרה לבין מערכת ההפעלה, ומעליו נמצאת התוכנית שלנו.

BIOS פועל בצורה דו-סיטרית: מצד אחד יש **פסיקות** (interrupts), שהן **בקשות** שירות שונות הנשלחות על ידי התוכנית שלנו, ומצד שני יוצר BIOS קשר עם התקני החומרה השונים למשל, דיסק כדי לטפל בבקשתנו.

כדי לקרוא לפסיקה של BIOS עלינו לטעון את האוגרים אשר ביע״מ במידע הנדרש כדי לבצע את השירות. זהו תהליך פשוט, המתבצע על ידי מספר פונקציות שבספריה dos.h.

הנה כך נראה קטע בתוכנית למילוי משימה זו:

```
union REGS
   {
      struct  WORDREGS  x;
      struct  BYTEREGS  h;
   };

struct BYTEREGS
   {
      unsigned char  al, ah, bl, bh;
      unsigned char  cl, ch, dl, dh;
   };
```

```
struct WORDREGS
   {
     unsigned int  ax, bx, cx, dx;
     unsigned int  si, di, cflag, flags;
   };
```

שים לב!

האיגוד REGS מכיל שני מבנים שכל אחד מהם מכיל את האוגרים של המעבד. נוכל להבחין במבנה BYTEREGS המכיל אוגרים בעלי 8 סיביות ובמבנה WORDREGS המכיל אוגרים בעלי 16 סיביות.

הצבת ערכים מתאימים באוגרים אלו והפעלת הפונקציה ()int86 יוצרים **פסיקת תוכנה** (software interrupt).

הפונקציה ()int86

הפונקציה ()int86 יוצרת ממשק פסיקות תוכנה המותאם למעבד 8086.

תבנית הפונקציה

```
int int86(int into, union REGS *inregs, union REGS
                                              *outregs);
```

נקודת ציון!

לפני ביצוע פסיקת תוכנה, הפונקציה מעתיקה את תוכן מבנה האוגרים אל האוגרים עצמם.

אחרי ביצוע הפסיקה, הפונקציה מחזירה את הערך של האוגר AX.

לצורך הדגמת כתיבת קוד היוצר פסיקה נבחר בפסיקת המסך ונערוך עמה היכרות קצרה.

בטבלה הבאה נוכל להבחין בפסיקת המסך 10h ובפונקציות הקשורות לפסיקה זו.

שים לב!

אין הבדל בין 10h לבין Ox10. זו פשוט דרך שונה לכתוב מספר הקסדצימלי. תוכנית C מבינה את Ox10 כמספר הקסדצימלי.

טבלה 17.1 : פסיקת המסך 10h

INT 10H: Video Services

The following is a list of the video services provided by ROM-BIOS.

00H set video mode	0dH read graphics pixel dot
01H set cursor size/shape	0eH write character
02H set cursor position	0fH read video mode
03H read cursor position	10H set palette
04H read light pen	11H character generator
05H select active display page	12H special functions
06H scroll window up	13H write string
07H scroll window down	14H load LCD font/map intensity bit
08H read character/attribute	14H load LCD font/map intensity bit
09H write character/attribute	15H get display parms (PC Convertible)
0aH write character	1AH Query/Set Display Comb Code
0bH select color palette/border	1BH Query functionality/status
0cH write graphics pixel dot	1CH Save/Restore video state

טבלה 17.2 : טבלת פונקציות לטיפול בסמן

01H set cursor size/shape (text modes). When visible, cursor always blinks.

Input: AH = 01H
CH = start line (0-1fH; 20H=no cursor)
CL = end line (0-1fH)

02H set cursor position. Hint: set past bottom line to make cursor invisible

Input: AH = 02H
BH = video page (0-based)
DH,DL = row,column (0-based)

03H read cursor position and size

Input: AH = 03H
BH = video page (0-based)
Output: DH,DL = current row,column of cursor
CH,CL = current start,end line of cursor

מטבלאות אלו אנו למדים שיש להתייחס לפסיקה עצמה, ולאוגרים המתאימים כדי להפעיל את פונקציית BIOS המתאימה.

במסגרת התוכנית הבאה נלמד כיצד ליצור פסיקת תוכנה. פסיקה זו תחביא את הסמן על ידי פונקציית BIOS שעליה ניתן ללמוד מהטבלאות.

```
/*****      inter.c  :קובץ תקליטור        *****/

/*  יצירת פסיקת תוכנה  */

#include <dos.h>

void off_cursor(void);

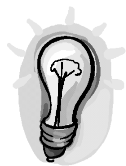
/* --------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; -----------------------------------------------------*/
int main(void)
 {
  off_cursor();
  return(0);
 }
/* --------------------------------------------------------
 ;  F U N C T I O N S
 ; -----------------------------------------------------*/
void off_cursor(void)
 {
  union REGS uRegs;
/*  הערך 1 באוגר AH מורה על התייחסות לגודל הסמן וצורתו
    ומתייחס לפונקציית BIOS היודעת להסתיר את הסמן.  */
  uRegs.h.ah = 1;
/*  הערך 0x20 מורה על הסתרת הסמן  */
  uRegs.h.ch = 0x20;
/*  הערך 7 מציין את סוף השורה  */
  uRegs.h.cl = 7;
/*  הפעלת פסיקת מסך 0x10 (10H)  */
  int86(0x10, &uRegs, &uRegs);
 }
```

הטבלה הבאה מפרטת את מערך הפסיקות ושירותי ROM BIOS. במסגרת ספר זה לא נפרט את הפונקציות המוכללות בכל פסיקה ופסיקה, כי קצרה היריעה לכך.

טבלה 17.3 : ***טבלת מערך הפסיקות ושירותי ROM BIOS***

Interrupts and ROM-BIOS Services

Interrupts	Services	Pointers
INT 00H Division by 0	INT 10H Video services	INT 1dH Video parms
INT 01H Single-step	INT 11H Equipment list	INT 1eH Diskette parms
INT 02H Non-Maskable	INT 12H Usable Memory Size	INT 1fH Graphics chars
INT 05H Print Screen	INT 03H Breakpoint	INT 13H Disk I/O
INT 06H (reserved)	INT 04H Overflow	INT 14H Serial Port I/O
INT 08H Timer	INT 15H AT Services	INT 41H hard disk
INT 0aH-0dH (hdwr ints)	INT 16H Keyboard I/O	INT 46H hard disk
INT 0eH Diskette	INT 07H (reserved)	INT 17H Printer I/O
	INT 18H ROM-BASIC	INT 43H EGA graphix chars
	INT 09H Keyboard	INT 19H Bootstrap
	INT 1aH Time I/O	INT 4aH user alarm addr
	INT 1bH Keyboard Break	INT 50H CMOS timer int
	INT 0fH (hdwr int)	INT 1cH User timer Int
	INT 20H-2fH DOS Interrupts	
	INT 33H Mouse Support	
	INT 67H Expanded Memory	



טיפ!

ניתן להפעיל פסיקה על ידי פקודת האסמבלר : int. נניח שאנו רוצים להפעיל פסיקה 5H (print screen). התוכנית הבאה תבצע זאת :

```
void main(void)
{
  asm int 0x5
}
```

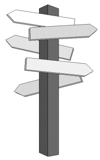
asm מציין למהדר C שלפניו כי מדובר בפקודת אסמבלר.

int פקודת ביצוע של פסיקה באסמבלר.

0x5 פסיקת **הדפסת מסך** (print screen).

הספריה bios.h

בספריה bios.h נמצא מספר פונקציות הפועלות ברמה נמוכה ופונות ישירות לפונקציות שנמצאות ב- ROM BIOS. פונקציות אלו מטפלות בעזרת פסיקות בתקשורת טורית, הדפסה, עבודה בדיסק, ועבודה במקלדת.



נקודת ציון!

*סדרת פונקציות אלו **אינן** שייכות לתקן ANSI.*

הפונקציה bioskey מטפלת ישירות בממשק המקלדת. תבנית הפונקציה:

```
int bioskey (int cmd);
```

הפונקציה מחזירה את הערך אשר הוקלד במקלדת. על הפרמטר cmd ניתן ללמוד מהטבלה הבאה.

*טבלה 17.4: **הפרמטר cmd***

cmd	תיאור		
0	ערך הקשה		
1	בדוק האם המקלדת מוכנה		
2	עבודה עם ה-shift סטטוס		
	הקשה	ערך	סיבית
	Right Shift	0x01	0
	Left Shift	0x02	1
	Ctrl	0x04	2
	Alt	0x08	3
	Scroll Lock on	0x10	4
	Num Lock on	0x20	5
	Caps on	0x40	6
	Insert on	0x80	7
2	עבודה עם ה-shift סטטוס 16bit		
	הקשה	ערך	סיבית
	Left Ctrl	0x0100	8
	Left Alt	0x0200	9
	Right Ctrl	0x0400	10
	Right Alt	0x0800	11
	Scroll Lock	0x1000	12
	Num Lock	0x2000	13
	Caps Lock	0x4000	14
	Sys Req	0x8000	15

הפונקציה bioskey ממתינה לבקשת מקלדת.

כיצד אם כך נוכל לקלוט הקשת מקש במקלדת, ולבצע פעילות מונחית מקשים? באמצעות התוכנית הבאה נלמד כיצד לקלוט מקש אחד מהמקלדת בעזרת הפונקציה bioskey(0).

```
        /*****     bioskey.c :קובץ תקליטור      *****/

                                          /* bios -קליטת מקשים דרך ה  */

#include <bios.h>
#include <stdio.h>
#include <stdlib.h>

                                                /* הגדרת ערכי מקשים  */
#define Right 0x4d00
#define Left 0x4b00
#define Esc 0x11b

void kbd(void);
int get_key(void);

/* ----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; -------------------------------------------------------*/
void main(void)
 {
  kbd();
 }
/* ----------------------------------------------------------
 ;  F U N C T I O N S
 ; -------------------------------------------------------*/
void kbd(void)
 {
  int key;

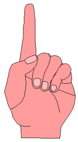
  while (( key = get_key() ) != Esc)
    {
        switch (key)
         {
         case Right : printf("Right\n");
          break;
         case Left  : printf("Left\n");
          break;
         }
    }
```

```
  if (key == Esc) exit(0);
  while (bioskey(1) == 0);
 }

int get_key(void)
 {
  int key, ok;

  do
   {
     key = bioskey(0);                        /* המתנה להקשת מקש */
     ok = (key == Right) || (key == Left) || (key == Esc);
     if(!ok) printf("key error, try again\n");
   }
  while(!ok);
  return(key);
 }
```

לאחר שלמדנו לקלוט מקש מהמקלדת, נתקלנו בבעיה קטנה: מהם הקודים של המקשים? גם נבדוק כיצד נוכל להשתמש בצירופי מקשים, כמו Ctrl+F1 למשל. תוכנית הדוגמה הבאה תלמד אותנו כיצד לקבל את קודי המקשים במקלדת, וגם צירופים שונים של מקשי פונקציה. מומלץ להעתיק קודים אלה על נייר ולאחר מכן להעבירם לקובץ הכרזות מסוים, נניח בשם keys.h, כדי שלא נבזבז את זמננו ב״גילוי מחדש״ של הקודים השונים.



שים לב!

לשליפת הקודים של צירופים מיוחדים, נעזרנו בטבלת הפרמטר cmd שבתחילת פרק זה.

```
/*****      קובץ תקליטור: getcode.c      *****/

/* שליפת הקודים של צירופי כל המקשים */

#include <bios.h>
#include <stdio.h>
#include <ctype.h>

#define RIGHT  0x01
#define LEFT   0x02
#define CTRL   0x04
#define ALT    0x08
```

```
/* ----------------------------------------------------------
 ;  M A I N - P R O G R A M
 ; ------------------------------------------------------*/
int main(void)
 {
   int key, modifiers;

                          /* הפונקציה מחזירה 0 כל עוד לא הוקלד מקש */
   while (bioskey(1) == 0)
     ;
                              /* הפונקציה מחזירה את המקש שהוקלד */
   key = bioskey(0);

                          /* הפונקציה קובעת את הצירופים המיוחדים */
   modifiers = bioskey(2);
   if (modifiers)
   {
        printf("[");
                          /* בודקים איזה  צרוף מיוחד דלוק אם בכלל */
        if (modifiers & RIGHT) printf("RIGHT");
        if (modifiers & LEFT)  printf("LEFT");
        if (modifiers & CTRL)  printf("CTRL");
        if (modifiers & ALT)   printf("ALT");
        printf("]");
   }
                                /* הצגת התווים/הקודים המתאימים */
   if (isalnum(key & 0xFF))
        printf("'%c'\n", key);
   else
        printf("%#02x\n", key);
   return 0;
 }
```

פרק 18

משימות

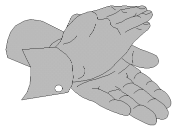
בפרק זה נתרגל מספר רב של נושאים, אשר הוצגו במהלך הלימוד בפרקי הספר הקודמים.

פרק 4

1. קובץ תקליטור: exr0401.c

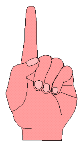
העצרת של מספר כלשהו n מחושבת על פי הנוסחה הבאה:

```
n! = 1 * 2 * 3 * ... * n
```



משימה:

כתוב תוכנית הקולטת את n, מחשבת ומדפיסה את העצרת שלו.



שים לב!

א. 0! = 1

ב. עצרת של מספר שלילי אינה חוקית.

ג. המחשב אינו מסוגל לחשב עצרת של מספרים מעל 170!

2. קובץ תקליטור: exr0402.c

מתמטיקאי איטלקי בשם פיבונצ'י, המציא וחקר סדרת מספרים הנקראת על שמו: **סדרת פיבונצ'י**. יש לה תכונות מעניינות, אשר לא נפרט כאן. הסדרה מורכבת כדלהלן: הערך של שני האיברים הראשונים הוא 1, וכל אחד משאר האיברים הוא סכום של שני האיברים הקודמים לו.

דוגמה למספרים הראשונים בסדרה זו:

```
1, 1, 2, 3, 5, 8, 13, ...
```

משימה:

כתוב תוכנית הקולטת את מספר האיברים מתוך סדרת פיבונצ'י, שיש לחשב ולהדפיס.

3. קובץ תקליטור: exr0403.c

ברצוננו להפעיל לולאות להדפסת צורות גיאומטריות שונות. לדוגמה, כדי להדפיס קו אופקי ניצור לולאה המדפיסה את התווים "*" זה אחר זה, ונקבל:

```
*****************
```

משימה:

כתוב תוכנית המדפיסה צורות גיאומטריות הבנויות מתווי "*":

א. ישר אופקי.

ב. ישר אנכי.

ג. מלבן מלא.

ד. משולש ישר-זווית.

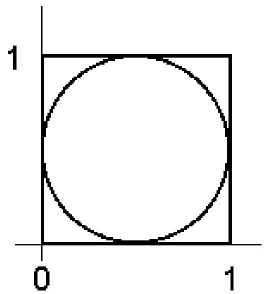
ה. משולש שווה-שוקיים.

ו. מעגל מלא (שים לב לכך שהמסך אינו סימטרי ב-x וב-y, ולכן צריך לתקן עיוות הגורם למעגל להיראות כאליפסה).

פרק 5

4. קובץ תקליטור: exr0501.c

הערך π כידוע, הוא מספר הפעמים בו נכנס קוטר של מעגל בהיקפו. בתרגיל זה נכתוב תוכנית לחישוב מקורב של הערך π. השיטה שנציג, מתבססת על שיקולים סטטיסטיים.



קיים ריבוע במידות 1*1 שבתוכו חסום מעגל בעל רדיוס 2/1 (ראה שרטוט). אנו "זורקים" לתוך הריבוע "אבנים קטנות" מספר רב של פעמים. נבדוק עתה כמה מהן נופלות בתוך שטח המעגל וכמה מחוץ לו, בשוליים שבין היקף המעגל להיקף הריבוע. אם נבצע מספיק זריקות, ותוך הנחה שפיזורן על פני שטח הריבוע אחיד, אפשר להסיק, שהיחס בין מספר ה"אבנים" הכללי שנזרק לבין מספר ה"אבנים" שנפלו בשטח המעגל - יהיה זהה ליחס שבין שטח המעגל לבין שטח הריבוע. כלומר,

$$\frac{\text{מספר אבנים שנפלו במעגל}}{\text{מספר אבנים כללי}} = \frac{\text{שטח המעגל}}{\text{שטח הריבוע}} = \frac{\pi * r^2}{1 * 1} = \frac{\pi * \left(\frac{1}{2}\right)^2}{1} = \pi * \frac{1}{4}$$

ומכאן,

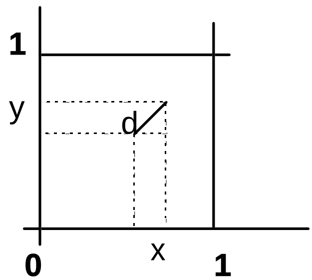
$$\pi = 4 * \frac{\text{מספר אבנים שנפלו במעגל}}{\text{מספר אבנים כללי}}$$

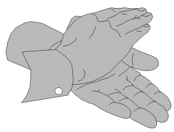
תוכנית המחשב שנכתוב צריכה לבצע סימולציה של "זריקת אבנים" לתוך הריבוע ולספור כמה מהן נפלו בתחום שטח המעגל. נשתמש לשם כך בפונקציה לייצור מספרים אקראיים, בשם ()rand, שנמצאת בספריה stdlib.h. קריאה לפונקציה זו פעמיים עבור כל זריקה, מאפשרת לקבל את ערך הנקודה (x, y), המבטא את המקום של נפילת "האבן שזרקנו".

מהאמור לעיל ניתן לסכם, ש"זריקה" פירושה חישוב נקודת נפילה (x ,y) אקראית באמצעות הפונקציה ()rand ונירמול התוצאות, כך שנקבל $0 \le x, y \le 1$.

מכיון שמרכז המעגל נמצא בקואורדינטות (0.5, 0.5), ניתן לחשב על פי נוסחת פיתגורס את מרחק נפילת ה"אבן" מהמרכז (ראה שרטוט). נקל לראות שאם D קטן, או שווה לרדיוס המעגל (2/1), אזי ה"אבן" נופלת בתוך שטח המעגל.

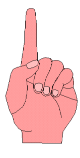
$$D = \sqrt{\left(x - \frac{1}{2}\right)^2 + \left(y - \frac{1}{2}\right)^2}$$





משימה:

כתוב תוכנית הקולטת את מספר הזריקות הרצוי ומחשבת בשיטה זו את ערכו המקורב של π.



שים לב!

בזמן הרצת התוכנית - ככל שמספר הזריקות רב יותר - כן תהיה התוצאה קרובה יותר לערך הנכון.

כדי להשתמש בפונקציה rand(), יש לרשום בתחילת התוכנית את ההוראה הבאה:

```
#include <stdlib.h>
```

לאחר מכן, ניתן לכתוב rand() = <משתנה> כדי לקבל מספר אקראי.

5. קובץ תקליטור: exr0502.c

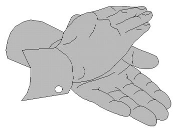
בתרגיל הקודם חישבנו את ערכו המקורב של הערך π באמצעים סטטיסטיים. בתרגיל זה נערוך חישוב מדויק יותר בטכניקה מתמטית הנקראת "פרישת טור לייבניץ" (על שם המתמטיקאי שהמציא שיטה זו). לא נפרט כאן כיצד ולמה פיתוח טור לייבניץ נותן ערך מדויק יותר של π, אלא נסתפק בהצגת הטכניקה, כדי שניתן יהיה להשתמש בה לצורך החישוב.

טור לייבניץ:

$$\frac{1}{4}\pi = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} = ... \pm \frac{1}{2*n-1}$$

כלומר, ערכו של π($\frac{1}{4}$) מתקבל על ידי הסיכום ההופכי של המספרים הטבעיים האי-זוגיים, תוך חילוף הסימן מחיובי (+) לשלילי (-) עבור כל איבר בטור.

כדי לקבל ערך מדויק של יש לסכם את האיברים עד נקודה מסוימת ולכפול ב-4. ככל שנסכם יותר איברים - כך תתקבל תוצאה טובה יותר. קיימת הוכחה מתמטית לכך שהשגיאה בתוצאה תהיה קטנה מהאיבר הבא, לאחר האיבר האחרון שחישבנו. כלומר, אם נסכם עד לאיבר החמישי שערכו 9/1, אזי השגיאה תהיה קטנה מהאיבר השישי, שערכו 11/1.



משימה:

כתוב תוכנית הקולטת את מספר האיברים שיש לסכם מתוך טור לייבניץ ומחשבת את ערכו של ואת ערך השגיאה המקסימלית.

פרק 6

6. קובץ תקליטור: exr0601.c

נתון מערך של שלמים (int) המכיל מספרים בסדר כלשהו, ונרשום את גודלו בקבוע MAX. נמיין את הערכים שבמערך בסדר עולה, מהקטן לגדול בשיטת **מיון בועות**.

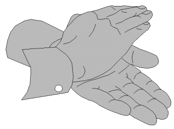
בתחילה, המערך הינו בלתי ממוין. נסרוק אותו מלמטה למעלה, כלומר מהאיבר האחרון, שהאינדקס שלו הוא MAX-1 כלפי איבר 0 (הראשון). בכל פעם נשווה שני ערכים צמודים, ואם הם בלתי ממוינים - נחליף ביניהם, ונמשיך בסריקה כלפי האיבר הראשון.

סריקה זו גורמה ל״הצפה״ של הערך הקטן ביותר עד לאינדקס 0, ולכן המערך נשאר במצב בלתי ממוין החל מאינדקס 1 (האיבר השני) עד לסופו.

נחזור על פעולת הסריקה שוב - מאינדקס MAX-1, אבל הפעם נמשיך רק עד אינדקס 1 (ולא 0). לאחר פעולה זו האיברים 0 ו-1 יהיו ממוינים. עלינו לחזור על הפעולה שוב - מסוף המערך ועד לאינדקס 2.

בכל סריקה אנו ״מציפים״ את הערך הקטן ביותר מבין האיברים הנותרים ושומרים ציון של האיבר האחרון הממוין. כאשר ציון זה יגיע לאיבר האחרון במערך (אינדקס MAX-1) - יהיה כל המערך ממוין כראוי.

פעולת ה״הצפה״ של הערכים הקטנים מזכירה בועות עולות במים, ומכאן נגזר שמה של השיטה.

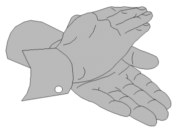


משימה:

כתוב תוכנית, שבה מוגדר הקבוע MAX. התוכנית קולטת MAX ערכים לתוך מערך שלמים בסדר כלשהו, ממיינת את המערך בשיטת הבועות, ומדפיסה את המערך הממוין.

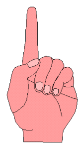
7. קובץ תקליטור: אין

ראה הסבר של שיטת מיון-בועות שבתרגיל הקודם.



משימה:

כתוב תוכנית דומה לתוכנית שבתרגיל הקודם. בתוכנית זו יבוצע המיון בשיטה הפוכה: סריקות חוזרות ונשנות של מערך השלמים, תוך כדי ״השקעת״ הערכים הגדולים כלפי מטה, אל עבר האיבר **האחרון**.



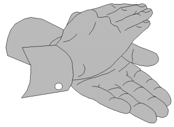
שים לב!

סדר הסריקה צריך להיות הפוך: מאינדקס 0 כלפי אינדקס MAX-1.

8. קובץ תקליטור: exr0602.c

בתרגיל זה נציג שיטת מיון המשלבת את טכניקת ״מיון-בועות״ בטכניקת ״השקעת הכבדים״. שיטה זו יעילה יותר משתי קודמותיה ונקראת ״מיון-הלוך-ושוב״. נשתמש בה כדי להציג מיון של מערך בסדר עולה.

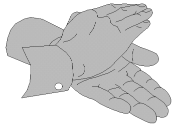
סורקים פעם אחת את מערך השלמים מלמעלה (ראש המערך) כלפי מטה (סוף המערך). בכל פעם שמגלים איבר לא מסודר מבצעים החלפה, ומיד לאחריה סורקים מנקודת ההחלפה כלפי אינדקס 0 בשיטת מיון-בועות. לאחר מכן, חוזרים לנקודה בה בוצעה ההחלפה וממשיכים בסריקה כלפי מטה. בשיטה זו שומרים את המערך ממוין עד לנקודה בה מגיעים. לכן, כשנסיים את הסריקה יהיה המערך כולו ממוין כפי שנדרש.



משימה:

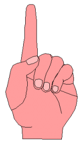
כתוב תוכנית, אשר קולטת ערכים בלתי ממוינים למערך שלמים, מבצעת מיון-הלוך-ושוב ומדפיסה את המערך הממוין.

9. קובץ תקליטור: אין



משימה:

כתוב תוכנית המבצעת מיון בסדר יורד של מערך שלמים בשיטת מיון-בועות: סדר הערכים יהיה מהגדול לקטן.



שים לב!

כדי לבצע מיון בשיטה זו, מתייחסים לבועות כאילו הן שוקעות ולא צפות.

משימה זו תוכל למלא על ידי ביצוע שינויים קלים בתוכנית שלתרגיל 6.

10. קובץ תקליטור: exr0603.c

תרגיל זה הינו משחק סימולציה של התפתחות אוכלוסיית "יצורים", אשר נקרא "**משחק החיים**" (The Game of Life). מחקרים רבים בביולוגיה מנצלים שיטות סימולציה כאלו לבדיקת תיאוריות שונות.

המשחק מוגדר כך:

קיים **מערך תווים** דו-מימדי, (char) המתאר תחום מחיה של יצורים. כל איבר במערך נקרא תא, והוא יכול להיות מאוכלס ביצור או להיות ריק. המשחק מתחיל בהגדרת מצב ראשוני במרחב המחיה כלומר, מי מהתאים אכן מאוכלס.

התוכנית מחשבת כיצד ייראה תחום המחיה בדורות הבאים, על פי מספר חוקים:

א. בתא מאוכלס היצור ימות מצפיפות, אם מספר שכניו גדול מ-3, או שהוא ימות מצער ושעמום אם מספר שכניו פחות מ-2.

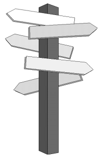
ב. בתא ריק ייוולד יצור חדש, אם מספר השכנים של התא הריק שווה בדיוק ל-2.

ניתן לשנות חוקים אלה ולהוסיף חוקים אחרים, בהתאם לרצונך. לדוגמה, התוכנית שבתקליטור מכילה מספר חוקים הנוגעים לפריצת מגיפה בקרב היצורים אם מספרם הכולל עולה על 200.

הגדרת השכנים של תא:

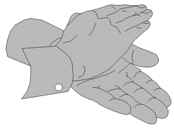
שכן 3	שכן 2	שכן 1
שכן 5	תא נבדק	שכן 4
שכן 8	שכן 7	שכן 6

נסמן בכוכבית ("*") תא המכיל יצור חי, ותא ריק נסמן בתו רווח (" ").



שים לב לנקודות הבאות!

כאשר התוכנית מחשבת דור חדש, יש צורך לתחזק שני מערכים: אחד מיועד לדור הקודם ואחד לדור החדש, כדי שתוצאות הדור החדש לא "יקלקלו" את ה"מצב" של הדור הקודם. בסיום חישוב הדור החדש, יש להעתיק את המערך אל מערך הדור הקודם.
אם מצב האכלוס הראשון הינו סימטרי, סביר שלפחות 20 הדורות הראשונים יוצגו בתמונות סימטריות מרהיבות עין (דוגמה מצורפת).



משימה:

כתוב תוכנית הקולטת את מצב האכלוס של הדור הראשון, מבצעת סימולציה של משחק החיים ומציגה את תחום המחיה של כל דור חדש.

ניתן לשכלל ולהוסיף חוקים כדי לבחון את השפעתם, ולאפשר למשתמש להקיש בכל פעם על כמה דורות לדלג כדי להציג את תמונת האכלוס.

```
Generation :1 citizens :24

            *****
            *****
            *****
             ***
             ***
             ***

Generation :9 citizens :7

           * * * *
          ** * * **
              *
        ***       ***
             ***
           **   **
       *    *   *    *
          * *   * *
        * **  *  ** *
      ** **  ***  ** **
      *   *       *   *
        *           *
         ****   ****
           * *** *
           *** ***
             * *
              *
```

```
Generation :11 citizens :105

            * * *
         *** * * ***
            ** **
       * *         * *
         *** *** ***
          **** ****
       ** * *   * * **
      ***           ***
        *     *     *
    *   *** *   * ***   *
    * * *  *     *  *  **
     * *      *      * *
       ******* *******

         ***  *  ***
           * * * *
             * *
             * *
```

11. קובץ תקליטור: exr0604.c

לעיתים יש צורך במהלך התוכנית לחפש איבר מסוים במערך, כדי לדעת אם הוא קיים או אינו קיים, ואם הוא קיים - היכן הוא נמצא במערך. כאשר מערך הנתונים ממוין, על פי סדר עולה או יורד, ניתן לבצע חיפוש כזה בצורה יעילה ביותר בטכניקה הנקראת "**חיפוש בינארי**". החיפוש הבינארי חוסך את הצורך לסרוק את המערך מתחילתו עד למציאת הערך (באם הוא קיים), או עד סופו (באם האיבר איננו קיים).

נוהל החיפוש הבינארי:

- תחילה בודקים את האיבר האמצעי במערך (ולכן צריך לדעת מראש את מספר האיברים במערך). כאן ייתכנו שלושה מקרים:

  א. זהו האיבר שמחפשים - במקרה זה סיימנו.

  ב. האיבר שמחפשים גדול בערכו מאיבר זה.

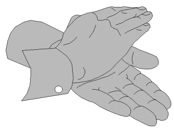
  ג. האיבר שמחפשים קטן בערכו מאיבר זה.

  במקרה ב' נסמן את אמצע המערך (האינדקס של האיבר שבדקנו) כסוף תת-מערך. במקרה ג' נסמן את אמצע המערך כתחילת תת-מערך.

- נחזור לשלב הבדיקה של אמצע תת-המערך המתקבל ושוב נבצע פעולת בדיקה וניתוח של שלושת המקרים האפשריים. באופן כזה, בכל שלב נקבל את האיבר הרצוי, או שנחלק את תת-המערך לחצי, ונחזור על הפעולה.

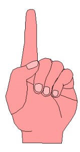
  כאשר גודל תת-המערך הנותר יהיה 1, נוכל לדעת בוודאות אם האיבר נמצא בו, או לא.

  מספר הבדיקות המקסימלי שנצטרך לבצע עבור מערך בגודל MAX איברים, יהיה: $\log_2 MAX$. זהו המספר המקסימלי של החלוקות לחצי שניתן לבצע. את מספר הבדיקות המקסימלי נעשה, כאשר האיבר הרצוי הינו האיבר האחרון שנבדק, או כאשר האיבר הרצוי אינו נמצא במערך.



משימה:

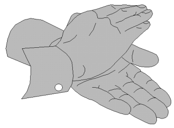
השתמש באחת התוכניות לקליטה ומיון של מערך שלמים לצורך הכנת מערך ממוין. הוסף קטע תוכנית הקולט מהמשתמש מספר שלם ומבצע חיפוש בינארי במערך הממוין. התוכנית תקבע אם האיבר קיים, או שאינו נמצא במערך. אם הוא נמצא, היא תדפיס את האינדקס שלו.



שים לב!

התוכנית צריכה לטפל במערך בעל מספר איברים זוגי או אי-זוגי.

12. קובץ תקליטור: אין



משימה:

כתוב את תוכנית החיפוש הבינארי (תרגיל 11) מחדש, כדי למצוא איבר במערך הממוין בסדר יורד. לשם הכנת מערך כזה השתמש באחת מתוכניות המיון שכתבת למיון מערך שלמים בסדר יורד.

פרק 11

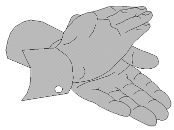
13. קובץ תקליטור: exr1101.c

נתון מערך של מחרוזות. גודלו נתון על ידי הקבוע MAX ואורך מחרוזת מקסימלי - על ידי הקבוע SLEN. ברצוננו למיין את מערך המחרוזות על פי סדר מילוני (לקסיקוגראפי).

כדי להשוות מחרוזות, קיימת בספריית <string.h> הפונקציה strcmp(), אשר דומה לפונקציה שהצגנו בפרק 11. פונקציה זו מקבלת שני מצביעים למחרוזות ומחזירה ערך השוואתי:

המחרוזת הראשונה גדולה מהשנייה	0<
המחרוזות זהות	0
המחרוזת הראשונה קטנה מהשנייה	0>

נשתמש בטכניקת מיון-בועות למיון מערך המחרוזות. לביצוע החלפה בין שתי מחרוזות יש להסתייע בפונקצית הספריה strcpy(), המעתיקה מחרוזת הנתונה על ידי המצביע השני ברשימת הארגומנטים שלה, אל המחרוזת הנתונה על ידי המצביע הראשון ברשימה. ראה בהמשך הסבר של הפונקציות לביצוע פעולות אלו.



משימה:

כתוב תוכנית הקולטת מחרוזות בלתי ממוינות לתוך מערך מחרוזות, מבצעת מיון-בועות ומדפיסה את המערך הממוין.

כדי להשתמש בפונקציות strcpy() ו-strcmp() יש לכלול בתחילת התוכנית את ההוראה:

```
#include <string.h>
```

אופן ההפעלה:

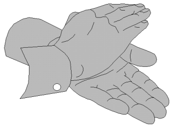
- ❖ השוואה בין מחרוזת s למחרוזת t:

```
strcmp(const char *s, const char *t)
```

- ❖ העתקת המחרוזת s לתוך t:

```
strcpy(char *const t, char *s)
```

14. קובץ תקליטור: אין



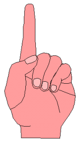
משימה:

כתוב תוכנית הדומה לתוכנית בתרגיל 13, כדי לבצע מיון לקסיקוגראפי יורד למערך מחרוזות.

15. קובץ תקליטור: exr1102.c

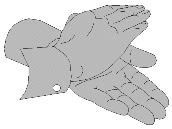
בשני התרגילים הקודמים התבקשת לבצע מיון-בועות של מערך מחרוזות. השיטה בה נקטנו לא היתה יעילה, כי כל פעולת החלפה בין מחרוזות גרמה למספר רב של הזזות נתונים בזיכרון המחשב. כדי להתגבר על בעיה זו, משתמשים בדרך-כלל **במיון-אינדקסים** למיון מערכי מחרוזות. נציג את השיטה:

מכינים מערך מקביל למערך המחרוזות, שבו יש בכל איבר מצביע לאחת המחרוזות שבמערך המחרוזות. מבצעים מיון-בועות של **מערך המצביעים**. הקריטריון לביצוע ההחלפה נעשה על ידי הפעלת הפונקציה ()strcmp על המחרוזות עצמן, אולם ההחלפה, במידה שישנה כזו, מבוצעת במערך המצביעים בלבד. בדרך זו חוסכים בהזזות של נתונים ומתקבל מערך מצביעים שכל איבר בו מצביע בסדר ממוין לעבר המחרוזות.



שים לב!

המחרוזות עצמן נשארות במערך המחרוזות בסדר המקורי, אולם אם נדפיס אותן באמצעות מערך המצביעים, יתקבל סדר המיון כנדרש.



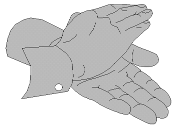
משימה:

כתוב תוכנית הקולטת מחרוזות לתוך מערך, ללא סדר מיון כלשהו. התוכנית תכין מערך מצביעים ותמיין אותו לפי הסדר הלקסיקוגראפי של המחרוזות. לאחר מכן, התוכנית תדפיס את המחרוזות לפי סדר המיון שבמערך המצביעים.

16. קובץ תקליטור: exr1103.c

נתון מערך מחרוזות בעברית, אשר יש למיין אותו בשיטת מיון-אינדקסים על פי סדר לקסיקוגראפי. כאן אנו נתקלים בבעיה טכנית הקשורה במיון תמליל בעברית: הפונקציה ()strcmp, בה אנו משתמשים לצורך השוואות, פועלת על המחרוזות משמאל לימין כמקובל באנגלית ובשפות אחרות. בשפה העברית יש צורך לבצע השוואה לקסיקוגראפית מימין לשמאל.

פתרון אחד לבעיה זו, הוא כתיבת פונקציה הנקראת ()mirror, אשר מקבלת מצביע למחרוזת בעברית והופכת אותה לתמונת ראי. לאחר מכן, ניתן להפעיל עליה את ()strcmp, ולאחר קבלת התוצאה - להפוך אותה חזרה למצב הראשוני באמצעות הפעלה נוספת של הפונקציה ()mirror.



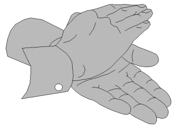
משימה:

כתוב תוכנית הקולטת מחרוזות בעברית לתוך מערך מחרוזות וממיינת אותו בשיטת מיון-אינדקסים, באמצעות הפונקציה mirror().

בעבודה ב-PC אנו מניחים שהפעלת את התמיכה HEBREW של יבמ, או תוכנה אחרת המיועדת לתמיכה בעברית, כמו HEB.

17. קובץ תקליטור: אין

בתרגיל הקודם תרגלת מיון אינדקסים של מחרוזות בעברית. פתרון נוסף לבעיית מיון מחרוזות בעברית הוא כתיבת מהדורה חדשה של הפונקציה strcmp(), כך שתפעל על מחרוזות בעברית. קרא לפונקציה זו hebcmp().



משימה:

כתוב את הפונקציה hebcmp() ובדוק אותה במצבים שונים.

שלב את hebcmp() בתוכנית של תרגיל 11 למיון אינדקסים של מחרוזות בלועזית. השתמש ב-hebcmp() בכל מקום בו השתמשת ב-strcmp(). כעת תקבל תוכנית למיון אינדקסים של מחרוזות בעברית!

18. קובץ תקליטור: exr1104.c

לפניך מספר נוסחאות ידועות לפיתוח בינום:

```
(a+b) = 1
(a+b) = a + b
(a+b) = a + 2ab + b
(a+b) = a + 3ab + 3ab + b
```

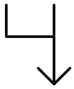
אם נסמן רק את המקדמים של האיברים, נקבל:

```
1
1 1
1 2 1
1 3 3 1
```

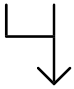
המשולש של מקדמי המשוואות קרוי **משולש פסקל**, על שם המתמטיקאי שחקר אותו. אחת התכונות המעניינות של משולש זה היא, שכל איבר בו מתקבל על ידי סיכום של שני איברים מהשורה הקודמת: האיבר שמעליו והאיבר הקודם לו.

ניתן לנצל תכונה זו לקבלת השורה הבאה במשולש פסקל:

```
1
1+1
```



```
1  2+1
```



```
1  3  3 1
```

נחשב את השורה הבאה:

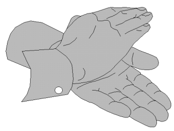
```
0+1=1; 1+3=4; 3+3=6; 3+1=4; 1+1=1     =>
```

נקבל:

```
   1 4 6 4 1
```

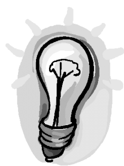
ואכן:

$$(a+b)^4 = a^4 + \underline{4}a^3b + \underline{6}a^2b^2 + \underline{4}ab^3 + b^4$$



משימה:

כתוב תוכנית הקולטת את מספר השורות שרוצים לבנות במשולש פסקל, מחשבת ומדפיסה את משולש פסקל עד השורה המבוקשת.

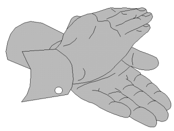


הדרכה:

התחל מהשורה הראשונה המכילה 1, כנקודת מוצא. השתמש בתכונה של משולש פסקל לחישוב כל שורה נוספת, כפי שהדגמנו.

ניתן לכתוב תוכנית זו בדרכים אחדות. התוכנית המצויה בתקליטור מתוחכמת למדי, ומטרתה ללמד שימושים מתקדמים בשפת C ומספר טכניקות תכנות מעניינות.

19. קובץ תקליטור: אין



משימה:

כתוב תוכנית להפקת משולש פסקל והדפסתו במבנה של משולש שווה-שוקיים, במקום משולש ישר-זווית. לדוגמה:

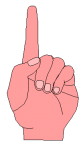
```
      1
    1   1
  1   2   1
1   3   3   1
```

20. קובץ תקליטור: exr1105.c

בתרגיל 18 הצגנו את הפיתוח הבינומי לנוסחה $(a+b)^4$. ברצוננו להסתייע בתוכנית לחישוב המקדמים של משולש פסקל, כדי לכתוב תוכנית המדפיסה את הפיתוח הבינומי המלא עבור ערך n נתון.

לדוגמה:

$(a+b)^3$ = 1*a^3 + 3*a^2b^1 + 3a^1b^2 + b^3

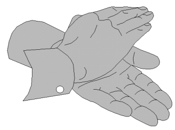


שים לב!

בכל איבר, סכום החזקות של a ו-b הוא n, כאשר האיבר הראשון הינו a וחזקת a פוחתת ב-1 בכל איבר.

הנוסחה הכללית לפיתוח הבינום היא:

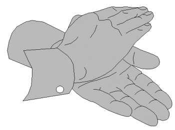
$$(a+b)^n = a^n + a^{n-1}*b^1 + a^{n-i}b^i + ... + b^n$$
i=0,1,...n



משימה:

כתוב תוכנית הקולטת את הערך n, מחשבת את השורה המתאימה ל-n במשולש פסקל ושומרת את המקדמים במערך. באמצעות מערך זה התוכנית מדפיסה את הפיתוח הבינומי המתאים.

21. קובץ תקליטור: exr1106.c



משימה:

כתוב תוכנית לחיפוש בינארי (ראה תרגילים 10 ו-11) במערך של מחרוזות הממוין במיון אינדקסים. לשם הכנת מערך כזה השתמש בתוכנית שכתבת למיון אינדקסים (ראה תרגיל 15).

22. קובץ תקליטור: exr1107.c

נגדיר פונקצית עזר לטיפול במחרוזות:

```
char *substr(char *s, int st, int len);
```

`char *s`	מחרוזת
`int st`	אינדקס התחלה של תת-מחרוזת
`int len`	אורך תת-מחרוזת

מטרת הפונקציה להחזיר מתוך המחרוזת s תת-מחרוזת המתחילה בתו המצוי באינדקס st ואורכה len תווים. הפונקציה צריכה לדעת לטפל במקרי קצה כמו למשל, כאשר st גדול מאורך המחרוזת s, או כאשר תת-המחרוזת ארוכה מאשר המחרוזת s. הפונקציה ()substr תחזיר את התוצאה במחרוזת s עצמה (כלומר ניתן "להרוס" את s המקורית). מותר להשתמש בפונקציית הספריה ()strlen, המחזירה אורך של מחרוזת. לדוגמה:

```
substr("ABCDEFG", 2, 4) -> "CDEF"
```

משימה:

כתוב את הפונקציה ()substr ושלב אותה בתוכנית ראשית ()main, הבודקת אם היא פועלת כראוי.

23. קובץ תקליטור: אין

עיין שוב בפרק 11, בהסבר על הפונקציה ()strcpy, המעתיקה מחרוזת למחרוזת. בספריית <string.h> קיימת פונקציה ()strncpy המעתיקה, בדומה ל-()strcpy, את המחרוזת t למחרוזת s. הפרמטר השלישי n מציין שהיא תעתיק לכל היותר n תווים.

הגדרת הפונקציה:

```
strncpy (char *s, const chat *t zise_tn);
```

משימה:

כתוב את הפונקציה ()strncpy ושלב אותה בתוכנית ראשית ()main, כדי לבדוק אם היא פועלת כראוי.

24. קובץ תקליטור: exr1108.c

משימה:

כתוב גירסה נוספת לפונקצית הספריה ()strstr, שתפקידה למצוא קיום של תת-מחרוזת במחרוזת.

הגדרת הפונקציה:

```
int pos(char *s, char *t);
```

כאשר s מצביעה על מחרוזת, שבתוכה צריך לחפש תת-מחרוזת המוצבעת על ידי t. הפונקציה מחזירה את מספר התו הראשון במקרה של מציאת תת-מחרוזת, או 1- אם לא קיימת כזו תת-מחרוזת.

25. קובץ תקליטור: אין

בשפה האנגלית כידוע, קיימים שני סוגי סדרים לאותיות: אותיות גדולות (Capitals) ואותיות קטנות. כאשר עורכים השוואה בין שתי מחרוזות ייתכן שנקבל ערך שקרי בגלל הבדל בסוג האותיות. לדוגמה, השוואת "aPPle", "apple" ו-"Apple" תיתן תוצאה שקרית בכל מקרה, למרות שלמעשה מחרוזות אלו מייצגות את אותה המילה.

כדי להתגבר על בעיה זו ניתן לתרגם את המחרוזות לסוג אותיות אחיד, לפני ביצוע ההשוואה. כלומר, להפוך את כל האותיות לגדולות או לקטנות.

בדיקה בטבלת ASCII מראה שההפרש בקוד ASCII בין אותיות גדולות לקטנות הינו 32, כאשר אותיות קטנות מקבלות את ערך ASCII הגדול יותר.

משימה:

כתוב שתי פונקציות המבצעות איחוד גודלי אותיות במחרוזת s.

הפיכת כל האותיות ב-s לגדולות	`toupper(s)`
הפיכת כל האותיות ב-s לקטנות	`tolower(s)`

שים לב!

ייתכנו במחרוזת תווים שאינם אותיות, ואותם יש להשאיר כמות שהם.

26. קובץ תקליטור: exr1109.c

משימה:

כתוב שתי פונקציות המתרגמות ערך הנמצא במשתנה מטיפוס char, למחרוזת הקסדצימלית ומחרוזת דצימלית, בהתאמה. משתמשים בפונקציות אלו כדי להדפיס (או להקרין) ערכים המצויים במשתנים. פעולתן דומה ופשוטה יותר מפעולת פונקציית הספריה printf().

הפונקציות יוגדרו כך:

```
ctoxs(char c, char *s);
```

הפיכת ערך המשתנה c למחרוזת הקסדצימלית שתישמר במצביע s, שנמסר על ידי המתכנת.

```
ctods(char c, char *s);
```

הפיכת ערך המשתנה c למחרוזת דצימלית שתישמר במצביע s, שנמסר על ידי המתכנת.

שים לב!

התוצאה המתקבלת עבור כל תו המצוי ב-c הינה למעשה ערך ASCII של התו.

27. קובץ תקליטור: exr1110.c

משימה:

כתוב שתי פונקציות, כדוגמת הפונקציות שבספריית המערכת של C, כדי לבצע את הפעולות הבאות:

א. תרגום מחרוזת ספרות למספר שלם (int):

```
atoi(const char *s);
```

הפונקציה מחזירה ערך שלם על פי מחרוזת הספרות המוצבעת על ידי s. היא מפסיקה את תהליך התרגום כאשר המחרוזת מסתיימת, או כאשר נתקלים בתו שאיננו ספרה.

ב. תרגום מספר שלם למחרוזת ספרות:

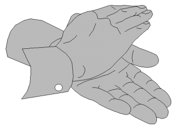
```
char *itoa (int i, char *s);
```

הפונקציה מתרגמת את הערך השלם שב-i למחרוזת ספרות מתאימה בתוך המצביע s שסופק, ומחזירה מצביע לתחילת המחרוזת הזו.

לדוגמה, פונקציה אשר תחזיר את הערך שב-i:

```
atoi (itoa (i, s));
```

28. קובץ תקליטור: אין



משימה:

כתוב פונקציה ()axtoi הדומה ל-()atoi, אשר מחזירה ערך שלם מתוך מחרוזת המכילה ספרות הקסדצימליות.

29. קובץ תקליטור: exr1111.c

בתרגיל זה נכתוב תוכנית מחשב המשחקת "בול-פגיעה" כנגד המשתמש. המחשב בוחר מספר בן 5 ספרות, על ידי הפעלת פונקציית הספריה ()rand, המחזירה מספר אקראי. המחשב דואג שספרה מסוימת תופיע פעם אחת לכל היותר. על המשתמש לנחש את הספרות, והמחשב בודק את מספר ה"בולים" (הספרות מופיעות במקום הנכון) ומספר ה"פגיעות" (הספרות הנכונות שאינן במקום הנכון).

למשתמש מוקצבים 10 נסיונות. אם הוא לא מצליח - התוכנית מדפיסה הודעת השתתפות בצער ומציגה את התוצאה הנכונה.

כדי שפונקציית ()rand תחזיר מספר אקראי שונה בכל הפעלה חדשה של התוכנית, צרף את השורות הבאות:

בתחילת התוכנית (קובץ המקור):

```
# include <time.h>
```

בתחילת ()main:

```
long now;
srand(time(&now)%37);
```

כדי שהפונקציה ()rand תחזיר ספרה בודדת (0-9) בכל פעם, כתוב:

```
digit = rand()%10;
```

30. קובץ תקליטור: exr1112.c

תרגיל זה מכוון לכתיבה של תוכנית המשחקת דמקה כנגד המשתמש. התרגיל מורכב יותר מכל התרגילים האחרים עד כה, אולם החלטנו לכלול אותו, מכיון שהוא מכיל מרכיבי תכנות מעניינים ביותר. אנו ממליצים בפני הקורא להתאמץ, להבין ולבצע את התרגיל.

כדי לפשט את הבעיה בשלב הראשון של לימוד העקרונות, הטלנו מגבלה על חוקי המשחק המקובלים:

בכל תור מותר להוריד כלי בודד בלבד. כלומר, לא ניתן לבצע הורדות מרובות, כפי שנהוג בדמקה. מגבלה זו חלה גם על המלך. המלך יכול להוריד באלכסון בכל הכיוונים, אולם עליו להתייצב במשבצת הצמודה מייד לאחר הכלי הראשון שהוא מוריד.

הנה הסבר קצר על הטכניקות בהן השתמשנו בכתיבת התוכנית שבתקליטור. עיון מדוקדק בתוכנית עצמה בוודאי יוסיף היבטים נוספים להבנת הנושא. תוכנית זו הינה "שחקנית" דמקה ברמה בסיסית, אשר הקורא מוזמן להכניס בה שינויים ושיפורים.

מבנה הנתונים בו בחרנו לייצוג מצב לוח הדמקה, הוא טבלה דו-מימדית בגודל 10*10. במרכזה מצוי לוח הדמקה במימדים 8*8 ומסביבו שוליים ברוחב אחד (בדומה לטבלה של "משחק החיים", ראה תרגיל 10).

בכל מקום בטבלה יכול להימצא אחד מתוך 6 ערכים:

-1	חייל מחשב
-2	מלך מחשב
1	חייל משתמש
2	מלך משתמש
0	מקום פנוי לנוע אליו
100	מקום שאסור לנוע אליו, כדוגמת השוליים או משבצות שאינן על אלכסונים מותרים.

התוכנית מכילה פונקציות בסיסיות לסידור ראשוני של הלוח והצגתו בכל שלב במשחק. ראה תרשים:

```
c>damka
   +---------------------+
1       X     X     X     X
2    X     X     X     X
3       X     X     X     X
4    .     .     .     .
5       .     .     .     .
6    0     0     0     0
7       0     0     0     0
8    0     0     0     0
   +---------------------+
    A  B  C  D  E  F  G  H
```

<-- מהו צעדך הבא ?

כאן מסומנים הכלים של המחשב ב-"X", הכלים של המשתמש ב-"0" והמקומות הפנויים ב-".". שים לב למספור השורות בספרות 1 עד 8 ומספור הטורים באותיות A עד H (בדומה לשחמט).

חלק אחד של התוכנית מוקדש להזזת הכלים של המשתמש על פי הנחיות שהוא מזין במקלדת. על המשתמש להקיש את זוג הקואורדינטות בו נמצא הכלי ואת זוג הקואורדינטות של היעד. על המחשב לבדוק את חוקיות התזוזה, או חוקיות ההורדה שהמשתמש מגדיר, לבצע עבורו את המהלך ולהציג את מצב הלוח החדש. ראה תרשים:

```
c>damka
   +---------------------+
1      X     X     X     X
2   X     X     X     X
3      X     X     X     X
4   .     .     .     .
5      .     .     .     .
6   0     0     0     0
7      0     0     0     0
8   0     0     0     0
   +---------------------+
    A  B  C  D  E  F  G  H
מהו צעדך הבא ? --> C6D5
   +---------------------+
1      X     X     X     X
2   X     X     X     X
3      X     X     X     X
4   .     .     .     .
5      .     0     .     .
6   0     .     0     0
7      0     0     0     0
8   0     0     0     0
   +---------------------+
    A  B  C  D  E  F  G  H
מהו צעדך הבא ? -->
```

חלק אחר של התוכנית נועד לבחירת מהלך המחשב. זהו החלק המורכב בבעיה. לצורך זה המחשב סורק את כל משבצות הלוח ומאתר את הכלים שלו. עבור כל כלי המחשב בודק את כל אפשרויות התזוזה.

חייל יכול לנוע צעד אחד קדימה ולהוריד צעד אחד בכל כיוון. מלך יכול לנוע על פני ארבעת האלכסונים, במידה ומסלול התנועה פנוי, ולהוריד כלי אחד בקצה כל מסלול כזה.

עבור כל תנועה אפשרית, התוכנית מחשבת ניקוד המביע את מידת ההצלחה של המהלך. תוך סריקת הלוח התוכנית זוכרת את המהלך הטוב ביותר שנמצא עד לאותו רגע, כדי שתוכל לבחור בו לשם ביצוע בסוף הסריקה.

לכל אחד מהקריטריונים שנבחרו נקבע ניקוד. במהלך הסריקה התוכנית בודקת את כל הקריטריונים וצוברת נקודות בהתאם. מצבים "טובים" למחשב הם בעלי ניקוד חיובי, ומצבים "גרועים" למחשב הם בעלי ניקוד שלילי.

הקריטריונים הנבדקים בתוכנית המצורפת הם:

א. תזוזת חייל קדימה מזכה בניקוד השווה למספר שורת היעד. כך נמשכים חיילי המחשב לעבר שורת ההמלכה (השורה הנגדית הקיצונית).

ב. תזוזת מלך אחורה מזכה ב-(<שורת היעד>-8). כך נמשכים מלכי המחשב חזרה לתוך לוח המשחק.

ג. הורדה של חייל אויב מזכה בנקודות.

ד. הורדה אפשרית של האויב בחזרה בעקבות המהלך הנבדק מפחיתה נקודות.

ה. מהלך של חייל הנמצא במצב סכנת הורדה מקבל נקודות דרבון כדי שיהיה לו כדאי לזוז ולהינצל.

ו. למהלך של חייל המגן על חייל אחר מפני הורדה מפחיתים נקודות, כדי לגרום לו להישאר במקום ולהמשיך להגן.

ז. מהלך של חייל, שלאחר ביצועו הוא יגן על חייל אחר, מזכה בנקודות נוספות כדי לדרבן אותו להגן על החייל החשוף.

ח. תזוזה של חייל לטורים הקיצוניים (טור A או טור H) מזכה בנקודות, שכן שם הוא מוגן יותר.

ט. תזוזה של חייל לשורה 8 מזכה בנקודות נוספות כדי להמריצו להיות למלך.

כדי לבדוק קריטריונים אלה, משתמשת התוכנית באלגוריתם הבא עבור כל כלי:

א. ייצור מהלך אפשרי וחוקי.

ב. ניקוד המהלך על פי נתונים גיאוגרפים (מספר שורה ומספר טור).

ג. ניקוד מקום הימצאות החייל לפני המהלך - האם מגן על חייל אחר, האם בסכנה.

ד. ניקוד כתוצאה מהורדה על ידי חייל אויב.

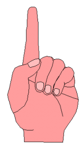
ה. ביצוע המהלך הנבדק באופן זמני.

ו. בדיקת הקריטריונים הנותרים לאחר ביצוע המהלך: האם כעת החייל מספק הגנה לחייל אחר, האם כעת הוא בסכנת הורדה.

ז. החזרת המהלך הזמני אחורה.

כאמור, המהלך שיבוצע למעשה יהיה זה שצבר את מירב הנקודות. התוכנית עורכת מחדש את מצב הלוח. ראה תרשים:

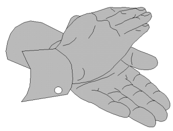
```
c>damka
  +----------------------+
1     X     X     X     X
2  X     X     X     X
3     .     .     .     X
4  X     X     0     .
5     .     0     0     X
6  0     .     .     0
7     0     0     .     0
8  0     0     0     0
  +----------------------+
   A  B  C  D  E  F  G  H
(C4) -> (E6) - 33
  +----------------------+
1     X     X     X     X
2  X     X     X     X
3     .     .     .     X
4  X     .     0     .
5     .     .     0     X
6  0     .     X     0
7     0     0     .     0
8  0     0     0     0
  +----------------------+
   A  B  C  D  E  F  G  H
(
```



שים לב!

בתוכנית המצורפת כללנו בפלט גם את הניקוד שקיבל המהלך הנבחר.

התוכנית בודקת לאחר כל מהלך אם קיים מצב ניצחון של אחד הצדדים, בכך שהוריד את כל החיילים של האויב. כמובן שבמקרה כזה המשחק מסתיים.

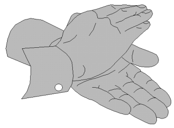


משימה:

על סמך ההנחיות שקיבלת, כתוב תוכנית דמקה משלך, והשווה אותה אל התוכנית הנמצאת בתקליטור המצורף.

אם לא תצליח בעצמך, למד בעיון את התוכנית המצורפת כדי להוסיף לה שיפורים במשימה הבאה.

31. קובץ תקליטור: אין



משימה:

שפר את תוכנית הדמקה שכתבת במסגרת המשימה הקודמת, או את זו המצויה בתקליטור המצורף, על פי הרעיונות הבאים ו/או רעיונות אישיים שלך:

א. הסרת המגבלה של הורדה בודדת בכל תור.

ב. ניתוח מעמיק יותר של תזוזות המלך, כך שאפשר יהיה לנצל אותו טוב יותר, כמו למשל להכנת מלכודות.

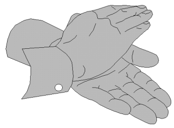
ג. שיפור הפונקציה הבודקת אם חייל נמצא בסכנת הורדה, כך שתבדוק גם סכנה ממלך אויב.

ד. בדיקה של יותר ממהלך אחד קדימה - משימה לא פשוטה!

ה. הוספת שיקולים של הקרבת חייל למען שיפור המצב. למשל, הקרבה של חייל, כך שאפשר יהיה להמליך מלך כתוצאה מכך.

32. קובץ תקליטור: אין

אחת הנקודות הקריטיות בהצלחת תוכנית משחק, כגון דמקה, היא קביעת הניקוד שמקבל כל קריטריון. שינוי הניקוד מניב אסטרטגיות משחק שונות.



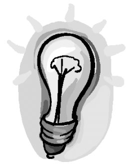
משימה:

שנה את תוכנית הדמקה, כך שהמחשב ישחק נגד עצמו, כלומר פעם יבחר המחשב מהלך עבור חיילי "X" ופעם עבור חיילי "O".

עבור כל סוג שחקנים, הנח למחשב להשתמש בניקוד שונה של הקריטריונים. על ידי עריכת משחקי מחשב-נגד-מחשב, תוכל לבחון אילו קריטריונים טובים יותר, ואותם לשלב בתוכנית הדמקה שלך.

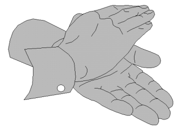
33. קובץ תקליטור: exr1113.c

במערכת ההפעלה UNIX קיימת תוכנית בשם "history". לאחר הפעלתה ניתן להמשיך להזין פקודות למערכת ההפעלה כרגיל. history צוברת את הפקודות המוזנות ב"מאגר היסטורי" ומאפשרת להפעיל פקודות שכבר הוקשו ושמורות במאגר בצורה מקוצרת, על ידי שימוש בתו "!", אשר ניתן להצמיד לו מספר פקודה או רצף אותיות שהן חלק מהפקודה. כאשר מזינים את הפקודה המיוחדת history, מופיעות על המסך כל הפקודות המצויות במאגר. התוכנית תפעל רק כאשר מבצעים הידור לקובץ EXE. כלומר, לא ניתן לביצוע עם התוכנה המצורפת Tclite.



טיפ!

במערכת DOS יש תוכנית דומה בשם "doskey".



משימה:

כתוב תוכנית history הפועלת בדומה לתוכנית history של UNIX. התייחס לכללים הבאים:

א. המאגר ההיסטורי יאפשר שמירת 30 הפקודות האחרונות שהוקשו.

ב. הקשת התו "!" תגרום להפעלה נוספת של הפקודה האחרונה שנשמרה במאגר.

ג. הקשת nn! (nn - מספר פקודה בתוך המאגר) תגרום להפעלת פקודה מספר nn מתוך המאגר.

ד. הקשת ccc...c! (ccc...c הינו רצף-תווים כלשהו) תגרום להפעלת הפקודה במאגר המתחילה ברצף התווים ccc...c.

לדוגמה, אם נקיש: di! תופעל הפקודה: dir *.c בהנחה, שקיימת פקודה כזו במאגר.

ה. הקשת הפקודה history תגרום להופעת הפקודות שבמאגר ובצמוד להן מספרם הסידורי.

ו. אם מקישים הוראה להפעלת פקודה מהמאגר ואין פקודה תואמת, תופיע על המסך הודעת שגיאה מתאימה.

לצורך כתיבת התוכנית היעזר בפקודות הספריה ()system, המוגדרת כך:

```
system(s)
  char *s
```

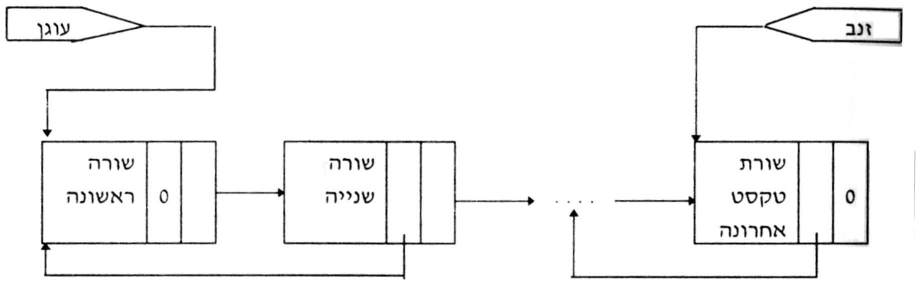
פונקציה זו מפעילה פקודה של מערכת ההפעלה, המצויה במחרוזת המוצבעת על ידי s, כאילו הפקודה הוקשה ישירות במקלדת.

התוכנית שתכתוב תקלוט פקודות לתוך מחרוזות ותפעיל פקודות נבחרות באמצעות ()system. עליה לבדוק תחילה אם הפקודה מתחילה ב-"!" או שהינה הפקודה "history".

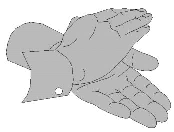
פרק 12

34. קובץ תקליטור: exr1201.c

אחת התכונות של תוכנה לעיבוד-תמלילים היא לאפשר למשתמש ל"נוע" על-פני שורות הטקסט במסמך לפנים ולאחור. מכיון שמספר השורות במסמך אינו ידוע מראש, יש להשתמש במבנה נתונים מסוג רשימה מקושרת כפולה. ברשימה זו, כל איבר יכיל שורת טקסט (מחרוזת) ושני מצביעים: אחד לאיבר הבא ברשימה, ואחד לאיבר הקודם ברשימה. באופן זה ניתן יהיה לנוע על-פני שורות הטקסט בשני הכיוונים.



העוגן מצביע לאיבר הראשון הזנב מצביע לאיבר האחרון.



משימה:

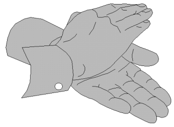
כתוב תוכנית מחשב, הקולטת באמצעות הפונקציה ()getch שורות טקסט (באורך מוגבל ל-60 תווים), ומאחסנת אותן ברשימה מקושרת כפולה. סיום הקלט יסומן על ידי שורה המכילה תווי רווח בלבד.

לאחר סיום הקלט, התוכנית תגיב בהצגת שורות הטקסט בזו אחר זו. תחילה על-פי סדר ההזנה שלהן ואחר כך בסדר הפוך לסדר ההזנה.

השתמש בפונקציות הספריה ()malloc כדי להקצות זיכרון לכל איבר חדש.

זכור להוסיף בתחילת התוכנית "include\alloc.h" include #.

35. קובץ תקליטור: אין



משימה:

בצע שינויים בתוכנית שבתרגיל 34 לבניית רשימה מקושרת כפולה, כך שבסיום הקלט יופיעו שורות הטקסט בסדר ההזנה שלהן ובצמוד לכל שורת טקסט יוצג מספרה הסידורי.

כדי לנהל את הרשימה, נשתמש בהוראות שונות להזנת שורות חדשות, ביטול שורות וסיום העיבוד. נשתמש לשם כך בתווי הבקרה I, D ו-Q:

Inn	הזנת שורה חדשה למקום כלשהו ברשימה. nn מציין את מספר השורה שלאחריה יש להוסיף את השורה החדשה.
Dnn	ביטול שורה מהרשימה. nn מציין את השורה שיש לבטל.
Q	סיום.

זכור לטפל במצבים השונים תוך התייחסות לעוגן ולזנב.

פרק 13

36. קובץ תקליטור: exr1301.c

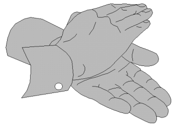
כפי שלמדנו, קיימת היררכיה בסוגי המשתנים, אשר מתבטאת בכך שהמספרים מהסדר הנמוך מוכלים בתוך המספרים מהסדר הגבוה:

- טיפוס char, המורכב מבית בודד ומכיל טווח מספרים 127 ±.
- טיפוס int, המורכב משני בתים ומכיל טווח 32565 ±.
- טיפוס long, המורכב מ-4 בתים ומכיל טווח 0x7FFFFFFFL ± (בהקסדצימלית) של ערכים שלמים.
- טיפוס float, המורכב מ-4 בתים ומכיל מספרים הכוללים נקודה עשרונית בטווח $3.37*10^{+38}$ ± → $8.84*10^{-37}$ ±.
- טיפוס double, המורכב מ-8 בתים ומכיל מספרים עם נקודה עשרונית בטווח $1.797*10^{+308}$ → $2.225*10^{-308}$ ±.

כאשר כותבים תוכנית מחשב, בוחרים למשתנים את סוג הטיפוס הרצוי על פי השימוש שאנו מייעדים לו, וטיפוס זה ידוע לכל אורך התוכנית. בשטחי תכנות מסוימים ובפרט באינטליגנציה מלאכותית, לא מטפלים בדרך כלל בקביעה של סוג הטיפוס של משתנה במהלך כתיבת התוכנית. לכן, כדי למנוע במקרים אלה תקלות בהמשך העבודה וההרצה, יש לאפשר למשתנה להיות מכל טיפוס שהוא. לשם כך יש לכתוב פונקציות כלליות מספיק, כדי שיידעו לטפל בצורה נכונה במשתנים בעלי טיפוס משתנה.

שפת C אינה מאפשרת (כחלק מהשפה) הגדרות כאלו, אולם ניתן באמצעות תכנות להגיע לתוצאה הרצויה תוך שימוש במבנה ה**איגוד** (Union).

בתרגיל זה נציג את הנושא ונאפשר לקורא להתמודד עם הבעיות שמתעוררות, להתוודע לאופן הטיפול בהן, ולאפשרויות השימוש בשיטה זו.



משימה:

הגדר באמצעות איגוד (Union) משתנה רב-טיפוסי המכיל:

`char ,int ,long ,float ,double`

כתוב את הפונקציות הבאות:

חיבור שני משתנים רב-טיפוסיים	`g_add()`
חיסור שני משתנים רב-טיפוסיים	`g_sub()`
הכפלת שני משתנים רב-טיפוסיים	`g_mul()`
חלוקת שני משתנים רב-טיפוסיים	`g_div()`

בכל מקרה, התוצאה המתקבלת אף היא משתנה רב-טיפוסי. כדי לראות את התוצאות, הכן פונקציה המדפיסה את ערכו של משתנה רב-טיפוסי בלווית הטיפוס שלו.

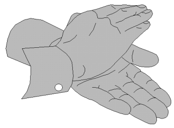
הפונקציות צריכות לטפל במצבים הבאים:

1. ביצוע הפעולה בין שני משתנים מטיפוסים שונים.
2. מעברים לטיפוסים גבוהים יותר שבהירכיה, כאשר הטווח דורש זאת, או כאשר הדיוק דורש זאת.

   לדוגמה, חלוקת שני שלמים (int) תיתן מספר עם נקודה עשרונית. אם יש שבר עשרוני בחלוקה. מכפלת שני מספרים שלמים (int) יכולה לתת מספר מסוג long, כאשר גודל התוצאה מחייב זאת.
3. מעברים להירכיה נמוכה יותר, כאשר הדבר מתאפשר.

   לדוגמה, משתנה מטיפוס float יהפוך למשתנה שלם, אם אינו מכיל שבר עשרוני.

37. קובץ תקליטור: אין

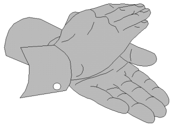


משימה:

שכלל את התרגיל הקודם, כדי שיכיל את הפונקציות הנוספות:

ערך מוחלט של משתנה רב-טיפוסי	`g_abs()`
ערך שלם של משתנה רב-טיפוסי	`g_floor()`
העלאת מספר רב-טיפוסי בחזקת מספר רב-טיפוסי	`g_pow()`

38. קובץ תקליטור: אין



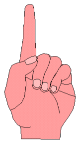
משימה:

עדכן את הפונקציות של שני התרגילים הקודמים, כדי שתפעלנה עבור טיפוס נוסף מסוג נקודה. **טיפוס נקודה** מורכב משני ערכים המגדירים שיעורי קואורדינטות במרחב דו-מימדי (קואורדינטות קרטזיות).

נקבל את טיפוס הנקודה, אם נוסיף שיעור שני של 0 לכל הטיפוסים הקודמים. כלומר, הערך מטיפוס float של 5.5 מוגדר במרחב הדו-מימדי כ-(5.5, 0).

בתרגיל זה נסתפק בעדכון הפונקציות האלו:

```
g_add() ,g_sub() ,g_abs() ,g_floor()
```



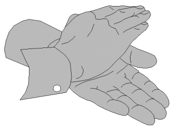
שים לב!

חיבור וחיסור של משתנים מטיפוס נקודה מתבצע לחוד על כל שיעור. כדי לחבר או לחסר משתנה מטיפוס נקודה במשתנה מטיפוס אחר, יש להפוך את המשתנה האחר למשתנה מטיפוס נקודה עם שיעור שני 0.

פרק 15

39. קובץ תקליטור: exr1501.c

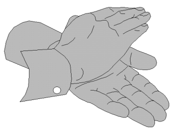
נתון מערך שלמים בלתי ממוין בעל מספר איברים המוגדר על ידי הקבוע MAX. נשתמש בטכניקה רקורסיבית: קריאה רקורסיבית של MAX-1 האיברים האחרונים (למעט האיבר הראשון), והכנסת האיבר הראשון למקום המתאים בתהליך של סריקה. בסיום נקבל מערך ממוין בסדר עולה.



משימה:

כתוב תוכנית, הקולטת ערכים לא ממוינים למערך שלמים בעל גודל MAX. התוכנית תבצע מיון רקורסיבי למערך ותדפיס אותו בתום המיון.

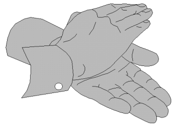
40. קובץ תקליטור: אין



משימה:

כתוב תוכנית, אשר דומה לתוכנית בתרגיל 39. בתוכנית זו תבצע מיון רקורסיבי של מערך שלמים בסדר יורד, במקום בסדר עולה.

41. קובץ תקליטור: אין



משימה:

(ראה הסבר לתרגיל 17).

כתוב תוכנית הקולטת את הערך n, מחשבת את ערך העצרת שלו באופן רקורסיבי, ומדפיסה את התוצאה.

42. קובץ תקליטור: exr1502.c

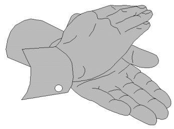
יש לחשב את סדרת פיבונצ׳י באופן רקורסיבי (ראה תרגיל 2). בדרך כלל, בתכנון פונקציות רקורסיביות מגדירים את משתני העזר כאוטומטיים, כדי שתתבצע הקצאה חדשה של משתנים בכל קריאה רקורסיבית. הדפסת סדרת פיבונצ׳י ברקורסיה הינה מקרה קלאסי, בו רצוי להגדיר את המשתנים כסטטיים, כדי שישמרו את ערכם בכל קריאה רקורסיבית.

הנוסחה הרקורסית של האיבר ה-n היא:

$$f_n = f_{n-2} + f_{n-1}$$

לכן, ניתן לכתוב פונקציה רקורסיבית, ללא משתנים סטטיים, לחישוב האיבר ה-n באמצעות שתי קריאות רקורסיביות עבור (n-2) ו-(n-1) - נסה זאת.

הפתרון הנתון בתקליטור מנצל אפשרות הגדרה של משתנים פנימיים סטטיים בפונקציה רקורסיבית. פתרון זה יעיל יותר.

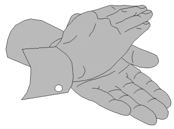


משימה:

כתוב תוכנית הקולטת את הערך n, מחשבת באופן רקורסיבי n איברים ראשונים של סדרת פיבונצ׳י ומדפיסה אותם.

43. קובץ תקליטור: exr1503.c

עיין בהסבר שבפרק 15 על עץ בינארי ממוין.



משימה:

כתוב תוכנית הקולטת מהמשתמש ערכים שלמים ובונה עץ בינארי ממוין. התוכנית תבצע הקצאת מבנים דינמית באמצעות פונקציית הספריה ()malloc, המקבלת כפרמטר את גודל המבנה ומחזירה מצביע למבנה שהוקצה.

תוך כדי בניית העץ, התוכנית תדפיס את המהלך על פני העץ, עד למציאת המקום המתאים להצבת האיבר החדש. בכל פנייה לאורך ענף שמאלי תודפס האות ״ש״, ובכל פנייה לאורך ענף ימני תודפס האות ״י״. ניתן יהיה לעקוב אחר התהליך של בניית העץ על ידי שרטוטו באמצעות נייר ועפרון.

כדי להשתמש ב-()malloc כתוב בתחילת התוכנית את ההוראה:

```
#include "include\alloc.h"
```

44. קובץ תקליטור: exr1504.c

עיין בהסבר על עץ בינארי, שנמצא בפרק 15.

משימה:

כתוב תוכנית המבצעת "טיול" על פני עץ בינארי ממוין, ומדפיסה את הערך של כל צומת ועלה על פי סדר מיון עולה. התוכנית תפעל ברקורסיה. לבניית העץ הבינארי היעזר בתוכנית של תרגיל 43.

45. קובץ תקליטור: אין

משימה:

עדכן את תוכנית הטיול וההדפסה של עץ בינארי ממוין, מהתרגיל הקודם, כדי שתדפיס את האיברים בעץ על פי סדר מספרי יורד.

46. קובץ תקליטור: אין

משימה:

כתוב מחדש את קטע התוכנית לבניית עץ בינארי ממוין (ראה תרגיל 43), כדי שתהליך הבנייה יתבצע ברקורסיה.

47. קובץ תקליטור: exr1505.c

משימה:

עדכן את התוכנית מתרגיל 44, המבצעת טיול והדפסת עץ בינארי ממוין, כדי שתדפיס רק את העלים של העץ.

זכור, כי העלים הם המבנים חסרי-בנים, בקצוותיהם של ענפי העץ.

פרק 16

48. קובץ תקליטור: exr1601.c

לעיתים ברצוננו לראות בצורה נוחה את תוכנו של קובץ כלשהו. אם קובץ הינו קובץ טקסט, ניתן להשתמש בפקודת type בסביבת העבודה של DOS כדי להציג את תוכנו על המסך. אולם, אם זהו קובץ נתונים המכיל ערכים בינאריים שאינו ניתן לייצוג כטקסט, פעולה זו תגרום קרוב לוודאי להשמעת צפצופים והצגת "פרצופים מחייכים" על המסך.

במקרה כזה יהיה נוח יותר להציג את תוכן הקובץ בצורה דצימלית, או הקסדצימלית. תוכנית השירות שעושה זאת נקראת במערכות רבות DUMP. בדרך כלל היא מציגה את תוכן הקובץ בקבוצות של 256 בתים ("סקטור"). היא משתמשת להצגת תוכן הקובץ בזוג ספרות הקסדצימליות לכל בית ובמקביל היא

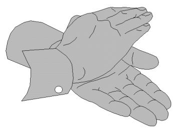
מציגה את התווים עצמם, אם קיימים. דוגמאות לכך ניתן לראות בהפעלת האופציה VIEW בתוכנת PCTools, או על ידי הפעלת תוכנית הדוגמה שבתקליטור המצורף.

```
   1    דף    exr1101.c    קובץ

0000000 : 23 69 6E 63 6C 75 64 65 20 22 69 6E 63 6C 75 64    #include "includ
0000016 : 65 5C 66 63 6E 74 6C 2E 68 22 0D 0A 23 69 6E 63    e\fcntl.h"..#inc
0000032 : 6C 75 64 65 20 22 69 6E 63 6C 75 64 65 5C 69 6F    lude "include\io
0000048 : 2E 68 22 0D 0A 0D 0A 0D 0A 0D 0A 63 68 61 72 20    .h"........char 
0000064 : 2A 63 74 6F 78 73 20 28 63 2C 20 73 29 0D 0A 20    *ctoxs (c, s).. 
0000080 : 20 20 63 68 61 72 20 63 2C 20 2A 73 20 3B 0D 0A      char c, *s ;..
0000096 : 7B 0D 0A 20 20 20 63 68 61 72 20 6C 75 74 5B 5D    {..   char lut[]
0000112 : 20 3D 20 22 30 31 32 33 34 35 36 37 38 39 41 42     = "0123456789AB
0000128 : 43 44 45 46 22 20 3B 0D 0A 20 20 20 63 68 61 72    CDEF" ;..   char
0000144 : 20 2A 73 61 76 20 3D 20 73 20 3B 0D 0A 20 20 20     *sav = s ;..   
0000160 : 69 6E 74 20 6F 66 66 73 65 74 20 3B 0D 0A 0D 0A    int offset ;....
0000176 : 20 20 20 6F 66 66 73 65 74 20 3D 20 28 63 20 26       offset = (c &
0000192 : 20 30 78 46 30 29 20 3E 3E 20 34 20 3B 0D 0A 20     0xF0) >> 4 ;.. 
0000208 : 20 20 2A 73 2B 2B 20 3D 20 6C 75 74 5B 6F 66 66      *s++ = lut[off
0000224 : 73 65 74 5D 20 3B 0D 0A 20 20 20 2A 73 2B 2B 20    set] ;..   *s++ 
0000240 : 3D 20 6C 75 74 5B 63 20 26 20 30 78 30 46 5D 20    = lut[c & 0x0F] 

   2    דף    exr1101.c    קובץ

0000256 : 3B 0D 0A 20 20 20 2A 73 20 3D 20 30 20 3B 0D 0A    ;..   *s = 0 ;..
0000272 : 0D 0A 20 20 20 72 65 74 75 72 6E 20 28 73 61 76    ..   return (sav
0000288 : 29 20 3B 0D 0A 7D 0D 0A 0D 0A 0D 0A 0D 0A 6D 61    ) ;..}........ma
0000304 : 69 6E 28 29 0D 0A 7B 0D 0A 20 20 20 63 68 61 72    in()..{..   char
0000320 : 20 66 69 6C 6E 61 6D 5B 32 30 5D 2C 20 6C 69 6E     filnam[20], lin
0000336 : 65 5B 31 38 5D 2C 20 62 75 66 5B 34 5D 20 3B 0D    e[18], buf[4] ;.
```

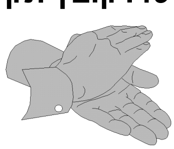


משימה:

כתוב תוכנית הקולטת מהמשתמש שם של קובץ ומבצעת Dump שלו על המסך. הצג קבוצות של 256 בתים בצורה הבאה:

16 שורות מכילות 16 בתים שכל אחד מהם מיוצג על ידי זוג ספרות הקסדצימליות. בהמשך השורה מוצגים אותם 16 הבתים כתווים (אם אפשר). לאחר הקרנת כל קבוצה תמתין התוכנית ללחיצת מקש לפני שתציג את הקבוצה הבאה.

49. קובץ תקליטור: אין



משימה:

בצע שינויים מתאימים במשימה הקודמת, כך שנתוני הקובץ יוצגו בספרות דצימליות במקום הקסדצימליות.

שים לב!

ערכו הדצימלי של בית הוא בתחום שבין 0 לבין 255, וערכו ההקסדצימלי הוא בתחום שבין 00 לבין FF. לכן, במשימה זו יש להציג כל בית באמצעות 3 ספרות דצימליות, במקום 2 ספרות הקסדצימליות.

50. קובץ תקליטור: exr1602.c

אחת הבעיות, שמתחבטים בהן לא מעט בעולם המחשבים, היא סודיות המידע והגנה עליו. נאמר, שקיים במחשב שברשותך קובץ נתונים, או תוכנית מחשב, שאינך מעוניין שלאחרים זולתך תהיה גישה אליהם.

אחת השיטות הנפוצות להגן על מאגר הנתונים, הנכס שלך או של הארגון, היא להצפינו על פי צופן הידוע רק לבעליו. הפעולות להצפנת נתונים ופענוח ההצפנה יכולות להיעשות בצורה פשוטה וממוכנת באמצעות תוכנית מחשב. גם שבירת צופן יכולה להיעשות באמצעות מחשב, אולם באפשרות זו לא נדון כאן. אנו נתייחס להצפנה פשוטה של קובץ נתונים כחסימת "פתח הקורא לגנב".

כדי להצפין קובץ נתונים, עלינו לפתח פונקציה המקבלת כארגומנט בית (או תו) ומשנה את ערכו באופן כזה, שהתהליך יהיה הפיך (כי עלינו גם לבצע פתיחת הצפנה) באופן חד-חד-ערכי. קיימות לכך שיטות רבות המוגבלות למעשה רק על ידי דמיונו של האדם.

בתוכנית הדוגמה המצורפת בתקליטור, בחרנו בשיטה פשוטה שנועדה להדגים עקרונות בסיסיים של הצפנה. אנו ממליצים שהקורא ישתמש בדוגמה זו כדי לפתח שיטת הצפנה "אישית" עבור הקבצים שלו.

משימה:

תכנן פונקציית הצפנה כפי שתואר בהסבר למעלה. הכן תוכנית מחשב הקולטת שם קובץ ומצפינה אותו באמצעות פונקציית ההצפנה.

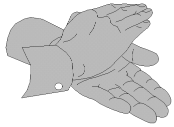
שים לב!

רצוי למחוק את הקובץ המקורי לאחר ביצוע ההצפנה, כי אחרת לא יהיה טעם בפעולה זו.

אזהרה!

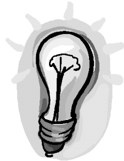
אל תמחק קובץ מקור לפני שהינך בטוח שפעולת ההצפנה בוצעה כראוי ושבידך הכלים לפתוח את ההצפנה. ועצה נוספת: שמור העתק של הקובץ המקורי בנפרד, במקום מוגן.

51. קובץ תקליטור: exr1603.c



משימה:

כתוב תוכנית מחשב הקולטת שם של קובץ שהוצפן באמצעות תוכנית ההצפנה של התרגיל הקודם, ומפענחת את ההצפנה, כדי לשחזר את הקובץ המקורי.



טיפ!

רצוי להשאיר את הקובץ המוצפן כפי שהוא, כדי שיהיה אפשר למחוק את הקובץ המשוחזר לאחר סיום העבודה איתו (אלא אם בוצעו בו שינויים), ולהשאיר את הגירסה המוצפנת בלבד. במקרה של שינוי הקובץ, יש להצפינו מחדש, כמובן.

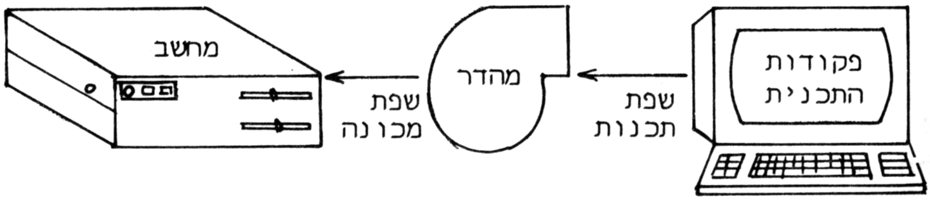
נספח א'

מבנה המחשב

הקורא המעוניין להרחיב את ידיעותיו בנושא, יוכל לעשות זאת באמצעות הספר **המדריך השלם לטכנאי PC - חומרה ותוכנה** מאת דורון סיון, **הוצאת הוד-עמי**.

שפת תכנות והידור

שפת תכנות הינה מתווך בין המתכנת לבין המחשב, והיא מאפשרת לו להורות למחשב מה ואיך לבצע. הוראות אלו מתורגמות לשפת מכונה, לשם הפעלת המחשב. תרגום ההוראות נעשה באמצעות **תוכנית מהדר** (Compiler), בתהליך הקרוי **הידור** (Compilation). לפניך תיאור סכימטי של התהליך בן שלושת השלבים :



*תרשים א.1 : **איך מתקבלת תוכנית בת-ביצוע***

לכל שפת תכנות ולכל מחשב קיים מהדר מיוחד להם, שכן המהדר חייב להכיר את שפת התכנות מחד, ואת שפת המכונה של אותו מחשב מאידך. תרגום ההוראות אינו חייב להיעשות מראש בתהליך הידור, אלא יכול להתבצע בעת ביצוע ההוראות, בזו אחר זו באמצעות **תוכנית מפרש** (Interpreter). דוגמה לכך הינה שפת BASIC.

הגדרות בסיסיות

- **סיבית** (Bit) - היחידה הקטנה ביותר לאחסון מידע, אשר מסוגלת לייצג שני מצבים בלבד: אפס ("0") ואחד ("1"), שחור ולבן וכו'.

- **בַּיִת** (Byte) - סידרה של 8 סיביות, אשר מייצגות בית אחד (לדוגמה אותיות, ספרות, סימני פסוק). בית מייצג למעשה **תו** (Character). כל פעולה המתבצעת במחשב על תו, מתבצעת בפועל על סדרות הסיביות האלו, ורק כאשר מבקשים מהמחשב להציג את המידע, נעשה "תרגום" לצורה הגרפית של התו שאנו מכירים. ראה בהמשך הצגת תווים במסך וטבלת ASCII לייצוג תווים במחשב. שים לב שתווי קוד ASCII מיוצגים בבית אחד.

- **מילה** (Word) - סידרה של 16 סיביות (2 בתים) המייצגות יחידת מידע גדולה יותר מבית אחד.

- **מילה כפולה** (Dword) - סידרה של 32 סיביות (4 בתים).

- **אוגר** (Register) - התקן שיכול לאחסן סדרת סיביות שלמה (בית או מילה). כאשר מאחסנים בתוכו סדרת סיביות, הוא מחזיק את ערכה, עד אשר משנים את תוכנו.

- **זיכרון** (Memory) - הזיכרון הינו התקן אלקטרוני המאחסן בתוכו מידע לוגי בצורה ספרתית, כלומר - סיביות. המידע יכול להיות הן תוכניות והן נתונים. מאפיינים עיקריים לרכיבי זיכרון הינם:

  - **כתיבה/קריאה או קריאה בלבד** (Read/Write or Read Only) - ניתן להבחין במספר סוגים של התקני זיכרון הנבדלים זה מזה בתכונה זו ובטכנולוגיה שבה הם בנויים.

  - **ROM/ERPOM** - התקני זיכרון המאפשרים פעולת קריאה בלבד.

  - **RAM סטטי/דינמי** (S-RAM/D-RAM) - התקן זיכרון המאפשר פעולות קריאה וכתיבה. המידע האגור ברכיבי זיכרון אלה נשמר כל עוד ההתקן מחובר למקור מתח.

  - **רוחב הזיכרון** (Memory Width) - מספר הסיביות שכותבים או קוראים לרכיב הזיכרון בפעולה אחת.

  - **גודל הזיכרון** (Memory Size) - מספר תאי הזיכרון, הבתים, ולמעשה - הכתובות, שקיימות לאחסון נתונים ותוכניות. גודל הזיכרון נמדד אם כן, במספר הבתים שהוא מכיל.

  - **זמן גישה** (Access Time) - משך הזמן הדרוש לזיכרון כדי לקבל, או למסור מידע מתא זיכרון (כתובת). ערך זה קובע אם הזיכרון מסוגל למסור או לקבל את הנתונים מהמעבד הראשי בקצב שבו המעבד פועל, או שהמעבד נדרש לפעול במהירות נמוכה יותר כאשר הוא ניגש לזיכרון. שימוש בזיכרון אשר זמן הגישה אליו ארוך יותר מהזמן הנדרש על ידי המעבד המרכזי לביצוע הפעולה, מאט את קצב המערכת ומקטין את ביצועי המערכת כולה.

❖ **מרחב זיכרון** - מושג זה מתייחס תמיד למרחב זיכרון RAM. המרחב הזה מוגדר בבתים בני 8 סיביות כל אחד, אשר את הראשון נסמן "בית 0".

❖ **מיפוי זיכרון** - בתהליך זה מחלקים את הזיכרון לקטעים שלכל אחד מהם מוגדר תפקיד, או ייעוד. כל פנייה של המעבד לקטע זיכרון זה תשמש לייעוד זה בלבד.

❖ **מקטע** (Segment) - אזור בזיכרון בגודל של 64KB. בתוך המעבד קיים אוגר מקטע המצביע על כתובת התחלתית של המקטע ומאפשר לו לגשת לכל כתובת במרחב של 64KB מכתובת הבסיס והלאה, בלי לשנות את ערכו של אוגר המקטע.

❖ **היסט** (Offset) - הערך שיש להוסיף לערך של "כתובת בסיס" כדי להגיע לכתובת הפיסית שבה מאוחסן נתון כלשהו.

❖ **מיעון ישיר** (Direct Addressing) - צורת גישה לזיכרון בפקודה מתוך אוסף הפקודות של המעבד.

❖ **מיעון עקיף** (Indirect Addressing) - צורת גישה לזיכרון בפקודה מתוך אוסף פקודות של המעבד. הכתובת שאליה מתייחסים לצורך הבאת נתון מזיכרון או כתיבה של נתון לזיכרון, איננה מוזכרת בתוך הפקודה עצמה. בפקודה מוזכר רק המקום בו נמצאת הכתובת, שאליה יש לרשום או שממנה יש לקרוא את הנתון.

❖ **מעבד, יחידת עיבוד מרכזית - יע"מ - CPU** (**C**entral **P**rocessing **U**nit) - התקן המכיל בתוכו רכיבים אחדים:

- **אוגרים** - משמשים לאחסון ביניים של נתונים, לשמירת מידע על מצב המערכת וניהול הכתובות.
- **יחידה אריתמטית/לוגית - ALU** (**A**ritemetic **L**ogic **U**nit) - מאפשרת להעביר נתונים בין האוגרים השונים ולבצע פעולות מתמטיות ולוגיות על נתונים שנמצאים באוגרים ובזיכרון.
- **יחידת בקרה** - מאפשרת לשלוט על העולם החיצוני, לקרוא נתונים ולרשום נתונים לעולם חיצוני זה.

❖ **מעבד עזר מתמטי** (Mathematic Co-Processor) - התקן עזר אשר מתחבר כתוספת למעבד (ובמעבד חדש - כחלק ממנו), ומאפשר לו לבצע פעולות מתמטיות על נתונים ממשיים (Floating point).

❖ **מרחב קלט/פלט** (I/O) - המעבד שולט על שני מרחבי כתובות - מרחב זיכרון ומרחב קלט/פלט. במרחב קלט/פלט ממופים התקני הקלט והפלט של המחשב כדי לאפשר למעבד גישה נוחה. כלומר, הוא דומה למרחב הזיכרון מבחינה זו שגם ממנו יכול המעבד לקרוא ולכתוב. ההבדל הוא שמרחב זה קטן בהרבה ממרחב הזיכרון, ואין בו מקטעים והיסטים (ראה להלן). לכן הגישה אליו נוחה ומהירה.

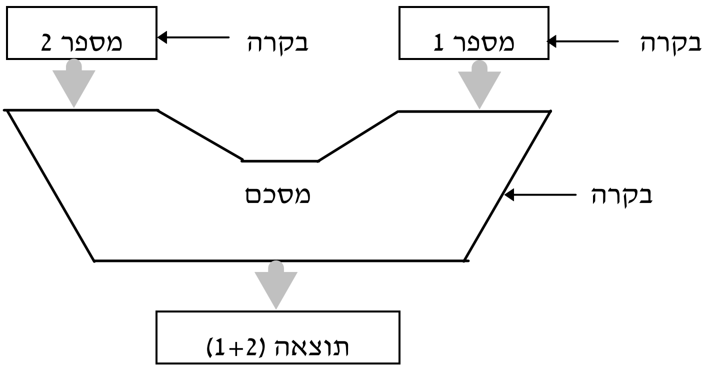
❖ **הבאת פקודה** (Fetch) - מחזור המכונה המאפשר הבאה של הפקודה ליחידת הבקרה כדי שהמעבד יוכל לבצע אותה.

יחידת העיבוד המרכזית - יע"מ - CPU

בפרק זה נתאר בצורה סכימטית מחשב **פשוט**, אשר מתאים למיקרו-מחשבים, מחשבים אישיים ואף לסוגים מסוימים של מחשבים קטנים (מיני-מחשבים). בהמשך, נלמד לבצע פעולות שונות באמצעות הוראות של שפת התכנות.

יחידת העיבוד המרכזית (Central Processing Unit) של המחשב מורכבת ממספר רכיבים, אשר לכל אחד מהם תפקיד מוגדר. הם קשורים ותלויים זה בזה, וביחד מהווים יחידה אחת מורכבת ומאוזנת.

המסכם - Adder



*תרשים א.2 : **המסכם***

יחידה זו מאפשרת למחשב לבצע פעולת חיבור בין שני מספרים ובאמצעותה ניתן גם לחסר מספר מחברו (על ידי חיבור מספר שלילי) ולכפול מספר במספר (על ידי חיבור חוזר של מספר לעצמו). קו בקרה "מאותת" למסכם כאשר מוכנים 2 מספרים בפתח המסכם. כאשר מסתיימת פעולת הסיכום, התוצאה מועברת הלאה.

האוגרים הכלליים - General Purpose Registers

האוגרים הפנימיים הם רכיבים אלקטרוניים המשמשים לאחסנת מספרים לביצוע חישובים שונים, ובהם מאוחסנות תוצאות הביניים של החישובים האלה. הם משמשים גם לאחסנה זמנית של נתונים בעת העברה ממקום למקום בזיכרון. יכולת הגישה הישירה לתוכן האוגרים, היא המקנה להם את תכונותיהם וחשיבותם. בנוסף לאוגרים הכלליים קיימים מספר אוגרים נוספים, בעלי תפקידים מיוחדים. את האוגרים במעבד מסמנים לפי תפקידיהם. כל האוגרים הינם בגודל 16 סיביות.

General Purpose Registers - אוגרים לשימוש כללי

AX	AH	AL	Accumulator	צובר
BX	BH	BL	Base	אוגר בסיס
CX	CH	CL	Count	מונה
DX	DH	DL	Data	נתונים

Special Purpose Registers - אוגרים ייעודים

SP	Stack Pointer	מצביע מחסנית
BP	Base register	מצביע בסיס
SI	Source Register	מצביע מקור
DI	Destination Register	מצביע יעד

FLAGS	Flags Register	אוגר דגלים

אוגר דגלים

אוגר הדגלים (Flag register) מכיל "דגלים" לשימוש התוכנית. דגלים אלה מושפעים מביצוע פעולות במעבד. על כן, התוכנית יכולה לבדוק את מצבם ולהגיב למצבים משתנים על ידי ביצוע קטע תוכנית שונה בהתאם למצב של כל אחד מהדגלים שבאוגר. מבנה האוגר:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	סיביות
				OF	DF	IF	TF	SF	ZF		AF		PF		CF	

CF -Carry **F**lag דגל הנשא.

PF - Parity **F**lag דגל הזוגיות.

AF - Auxiliary **F**lag דגל נשא עזר.

ZF - Zero **F**lag דגל אפס.

SF - Sign **F**lag דגל סימן.

TF - Trap **F**lag דגל מלכודת.

IF - Interrupt Enable **F**lag דגל פסיקה.

DF - Direction **F**lag דגל כיוון.

OF - Overflow **F**lag דגל גלישה.

אוגר מצב - Condition Code Register

CC	Z	N	C	P	V

אוגר זה קשור למסכם. לאחר כל פעולת סיכום הוא מתאר בכל אחד מהמדורים שלו את התכונות של התוצאה שהתקבלה בעזרת ערכי 0 או 1. מדור Z מציין על ידי "1" אם תוצאת הסיכום הייתה אפס (Zero). זוהי שיטה חסכונית במיוחד, כי בדיקת תוצאת המסכם הינה בדיקת מדור בודד. דוגמה לניצול תכונה זו תהיה בדיקת שוויון בין זוג מספרים כלשהו: במקום להשוות כל ספרה בשני המספרים, נחסר מספר אחד מחברו ונבדוק את מדור Z. אם המספרים היו שווים, תוצאת החיסור תהיה אפס וערכו של Z יהיה "1". תא N למשל, מסמן על ידי "1" אם תוצאת המסכם הייתה שלילית (Negative).

אוגר הפקודה - Instruction Register

נציג לדוגמה מחשב, שהפקודות בו נראות כך:

<אוגר נתונים> <אוגר יעד> <קוד הפעולה>

IR	OP CODE	target register (1)	data register (2)

פעילות המחשב מונחית על ידי פקודות בשפת מכונה, אשר הוא מבצע אותן זו אחר זו לפי הסדר המוכתב על ידי התוכנית. לפני הביצוע, נטענת הפקודה לתוך אוגר הפקודה. במחשב הדמיוני שלנו הוא מחולק לשלושה שדות: בשדה השמאלי מופיע **קוד הפעולה** (OP Code). זהו מספר המציין למחשב מהי הפקודה. שני השדות הנוספים מכילים בדרך כלל מספרי אוגרים כלליים המשתתפים במהלך ביצוע הפקודה.

אם נניח שהקוד לביצוע פעולת סיכום (ADD) הינו 14, וברצוננו לסכם תוכן אוגר 3 עם אוגר 1 - תהיה הפקודה בשפת מכונה:

OP CODE	1	2
14	3	1

במחשב שלנו, תוצאת הסיכום תוכנס לאוגר "הראשון" - ובמקרה זה אוגר מס' 3. אם ברצוננו להעתיק תוכן אוגר 2 לאוגר 4, ובהנחה שקוד פקודת ההעתקה המתאימה הינו 45, נקבל:

OP CODE	1	2
45	4	2

מונה הפקודות - Program Counter

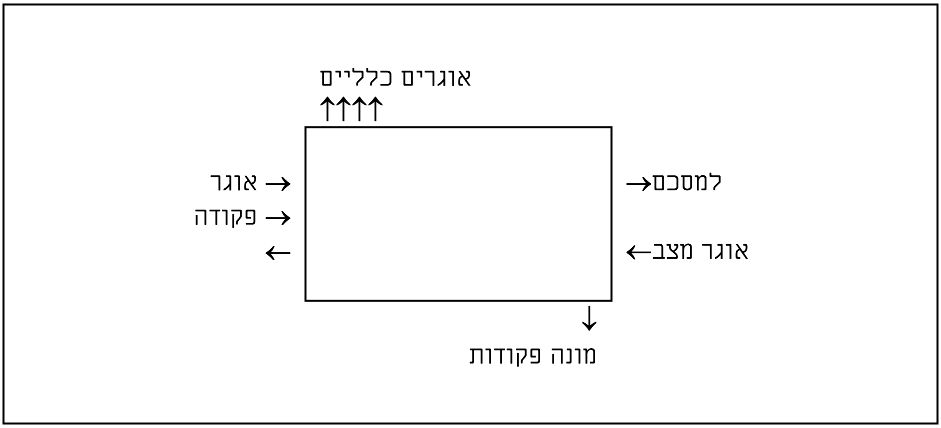


תוכנית מחשב מורכבת מרצף של פקודות, אשר נמצאות במערך הזיכרון שלו. לאחר סיום ביצוע הפקודה שבאוגר הפקודה, ממשיך המחשב באופן אוטומטי לפקודה הבאה העוקבת בזיכרון, ומעביר אותה לאוגר הפקודה לביצוע. מונה הפקודות מאפשר למחשב לעקוב אחר רצף הפקודות ומציין מהי הפקודה הבאה בתור לביצוע. הערך המצוי בו גדל באופן אוטומטי מיד לאחר העברת פקודה לאוגר הפקודה.

באמצעות מונה הפקודות מיישמים את מנגנון הדילוג, אשר מאפשר את הגמישות והתחכום של שפת המחשב על ידי דילוג ממקום אחד בתוכנית למקום אחר, כדי לבצע חלקים שונים שלה על פי תנאים מסוימים. הדילוג בפועל נעשה על ידי העברת המיקום החדש (כתובת תא הזיכרון) את מונה הפקודות, במקום כתיבת הפקודה הנמצאת בו ומיועדת לביצוע.

בקרה - Control

במרכז הפעילות שתוארה עד כה נמצאת פעולת הבקרה. תפקידה לכוון, לאותת ולתזמן את פעולת הרכיבים השונים במחשב. ביחידת הבקרה נמצא ה**מפרש/מפענח** (Interpreter) - רכיב שאחראי לפענוח קוד הפקודה שבאוגר הפקודה ותרגומו לרצף של אותות בקרה המחוללים את ביצוע הפקודה בפועל.



תרשים א.3 : הבקרה - צומת העצבים במעבד

ניהול הזיכרון, ולמעשה הגישה לזיכרון המחשב הינה באמצעות קבוצת **אוגרי מקטע**, אשר נציג להלן. הסבר על פעולתם יינתן בהמשך, בסעיף הדן בזיכרון.

אוגרי מקטע (מצביעי כתובת) - Segment Registers

CS	Code Segment	מצביע מקטע תוכנית
DS	Data Segment	מצביע מקטע נתונים
SS	Stack Segment	מצביע מקטע מחסנית
ES	Extra Segment	מצביע מקטע נוסף
IP	Instruction Pointer	מצביע הוראות

פסיקות (Interrupts)

כאשר המעבד עסוק בביצוע תוכנית מסוימת, אין באפשרותו לבדוק כל הזמן אם התקן מסוים מבין כל ההתקנים המחוברים אליו מבקש ממנו שירות כלשהו. כדי לקלוט בקשות שירות מההתקנים שמחוץ ליע״מ, משתמש המעבד במנגנון **הפסיקה** (Interrupt). זוהי ״הפרעה״ למעבד במהלך הפעילות על ידי סימון מוסכם. המעבד מפסיק (ומכאן המונח ״פסיקה״) את הפעילות שבה עסק בזמן שזיהה את בקשת הפסיקה ופונה לטפל בבקשה. עליו לשמור את מצב האוגרים ושאר הערכים הדרושים, כדי שיוכל לחזור בצורה תקינה ואמינה לפעילות שהופסקה. האוגרים הנשמרים אוטומטית הינם CS ו-IP, המצביעים על ההוראה הבאה שהמעבד יבצע בסיום הטיפול בפסיקה. גם אוגר הדגלים נשמר.

את הפסיקות ניתן לחלק לשני סוגים: **פסיקות חומרה** (Hardware Interrupts) **ופסיקות תוכנה** (Software Interrupts). פסיקות חומרה נחלקות לשני סוגים: **פסיקות שניתנות למיסוך** (Maskable) **ופסיקות שאינן ניתנות למיסוך** (Non Maskable). במשפחת המעבדים 86 ניתן להגדיר עד 256 פסיקות שמספרן 00-FFh.

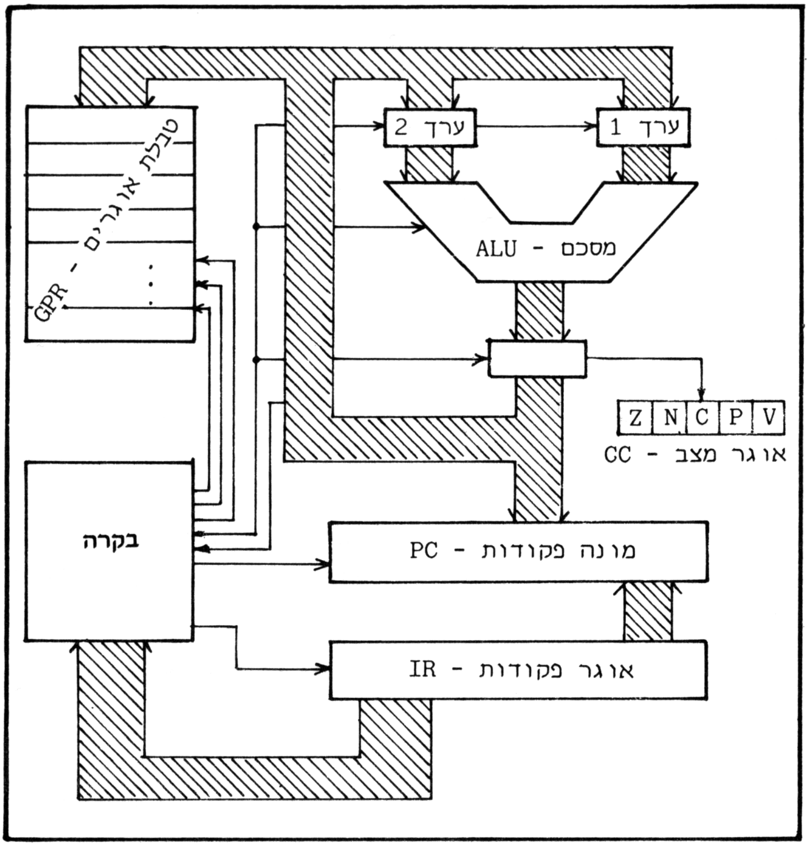
השלם מורכב מחלקיו - היע"מ

יחידת העיבוד המרכזית (היע״מ) הינה מכלול המורכב מהיחידות שהכרנו, ובנוסף - אוגר זיכרון. לעיתים כוללים בה גם את הזיכרון. עוצמת המחשב ויכולתו תלויים רבות בעוצמת היע״מ והמהירות בה היא מבצעת את הפקודות.

לפניך שרטוט סכמטי שמכיל את הקשרים ויחסי הגומלין בין הרכיבים השונים. הקווים הכהים בשרטוט מייצגים **אפיקי נתונים**, או **אפיקי מידע** (Data Buses, או Data-Paths), אשר דרכם מועברים ערכים מספריים בין רכיבי היע״מ. הקווים הדקים הינם **אפיקי הבקרה**, או **קווי הבקרה** (Control Buses, או Control Lines) המאותתים לרכיבים השונים מתי להיכנס לפעולה.

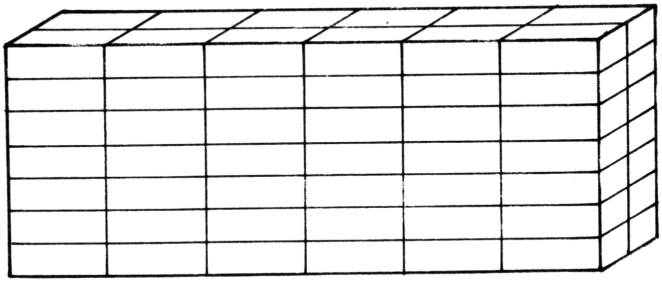
בין טבלת האוגרים הכלליים למסכם קיים אפיק נתונים המאפשר להעביר אל המסכם שני ערכים המאוחסנים באוגרים. תוצאת הסיכום מועברת לאחד האוגרים באמצעות אפיק המידע שנמצא ביציאת המסכם. בין אוגר הפקודה למונה הפקודות מצוי אפיק

מידע המאפשר טעינת מספר פקודה ישירות מפקודת דילוג. אפיק המידע בין אוגר הפקודה ליחידת הבקרה מאפשר מעבר קוד הפקודה למפענח הנמצא ברכיב הבקרה.



תרשים א.4 : **סכימה של היע"מ**

מערך הזיכרון



תרשים א.5 : **הזיכרון**

היע״מ בלבד אינה מספיקה כדי שהמחשב, ולמעשה - מערכת המחשב - תפעל כראוי, כי היא אינה יכול להחזיק את התוכנית הפועלת וגם את אוספי הנתונים הדרושים לעיבוד. היכן, אם כן, שומרים את התוכנית, הנתונים ותוצאות החישובים? התשובה לכך היא, **מערך הזיכרון** (Memory או Internal Storage). המחשב יכול לטעון אל היע״מ ערכים מתוך תאי הזיכרון. הם דומים במבנה לאוגרים הכלליים שביע״מ, אך אין קשר ישיר ביניהם לבין המסכם, או לשאר חלקי היע״מ.

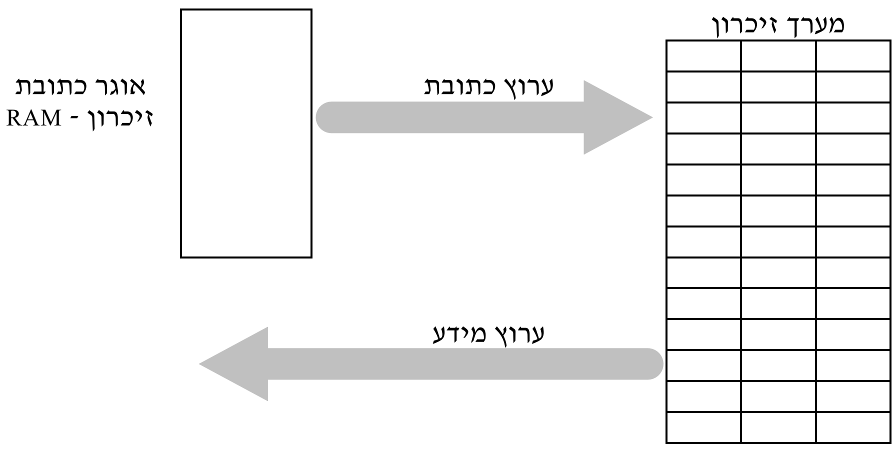
כדי לבצע פעולה בין שני ערכים בזיכרון, יש צורך לטעון את שניהם לתוך אוגרים ביע״מ ואז לבצע את הפעולה. התוצאה מועברת מאוגר התוצאה לזיכרון. כאשר מציגים למחשב פקודת BASIC כמו זו, לדוגמה:

```
LET  A = B + C
```

A, B ו-C הם שמות סמליים (כתובות) של תאים בזיכרון. ביצוע פקודה זו מתבטא בטעינת ערכי B ו-C לאוגרים ביע״מ, ביצוע הסיכום והעברת הערך שבאוגר התוצאה אל הזיכרון.

כיצד מנתב המחשב את הערכים מהזיכרון ליע״מ ולהיפך? לכל תא זיכרון יש **כתובת** (Address), והמחשב ״פונה״ אליו על ידי ציון כתובתו, שהיא מספרו הסידורי. לדוגמה: ספירת הכתובות במחשב מתחילה בערך אפס (0). לכן, כדי ״לפנות״ לתא 10,000, צריך לציין למחשב את הכתובת 9,999. המספור הזה הוא לפי ״**בסיס 0**״ (Zero Base), והוא אופייני לטיפול במערכי זיכרון. נפגוש בו רבות כשנעסוק בשפת C.

אוגר כתובת זיכרון - Memory Address Register



*תרשים א.6: **אוגר כתובות הזיכרון***

ניהול הפניות לכתובת בזיכרון מתבצע באמצעות אוגר הנמצא ביע"מ - **אוגר כתובת הזיכרון**. כדי לפנות לתא מסוים בזיכרון, המחשב טוען לאוגר זה את כתובת התא. הקשר בין מערך הזיכרון ליע"מ מתבצע באמצעות שני **אפיקים** (Buses). **אפיק הכתובות** ("Address Bus") - מוליך את הכתובת הנמצאת באוגר הכתובות למערך הזיכרון; **ואפיק הנתונים** ("Data Bus") - מוליך את הערכים בין האוגרים לבין תאי הזיכרון. כבר הזכרנו שמונה הפקודות ביע"מ מציין את מספר הפקודה הבאה שתיטען לתוך אוגר הפקודה. למעשה, טוען בו היע"מ את כתובת הפקודה.

מערך הזיכרון של המחשב מכיל את פקודות תוכנית המחשב ואת הערכים (הנתונים) המשתתפים בחישובים. כאשר המחשב מוכן לביצוע פקודה נוספת, מעלה היע"מ על אפיק הכתובות את תוכן מונה הפקודות (המכיל כתובת של פקודה) ועל אפיק הנתונים המידע הוא מעביר את תוכן הפקודה מתא הזיכרון לאוגר הפקודה. לפעולה כזו קוראים **מחזור זיכרון** (Memory Cycle).

המעבד מספק **אפיק כתובות** (Address Bus) של 20 סיביות ובכך הוא מאפשר גישה למרחב זיכרון של 1 מגה-בתים (2^{20}). זה הוא תחום כתובות הזיכרון (בייצוג הקסדצימלי): 000000-0FFFFFh. המעבד רואה את הזיכרון כ**רצף אחד** (Bank) של מיליון בתים, אשר לכל בית ברצף זה יש כתובת המגדירה אותו.

התוכניות המתבצעות במעבד רואות את הזיכרון בצורה לוגית כמחולק למקטעים בני 64KB, אשר השליטה בהם הינה על ידי אוגר מקטע ברוחב של 16 סיביות. כל מקטע כזה יכול להימצא בכל כתובת שהיא במרחב הזיכרון. כל כתובת במרחב הזיכרון מחושבת באופן יחסי לאוגר המקטע בהתאם לסוג הפעולה. כך למשל, הוראות המתייחסות לתוכנית תשתמשנה באוגר מקטע CS, הוראות המתייחסות למחסנית תשתמשנה באוגר מקטע SS וכו'.

המעבד יכול לאחסן **מילה** (word) של 16 סיביות בזיכרון בכתובת זוגית או אי-זוגית לחילופין, ללא מגבלה כלשהי. קיימת חשיבות לאחסון מילים בכתובות זוגיות, כי הדבר מאפשר למעבד לקרוא נתון ברוחב 16 סיביות בגישה אחת לאפיק הנתונים במחשבים החדשים הגישה היא ל-32 סיביות. בגלל מבנה החומרה אחסון הנתון בכתובת אי-זוגית יחייב שתי גישות של המעבד לאפיק הנתונים ויאריך את זמן הביצוע של ההוראה.

נראה עתה כיצד משתמש המעבד באוגרי המקטע לחישוב של כתובת אמיתית. כתובת אמיתית במרחב של 1MB מורכבת משני אוגרים של 16 סיביות: **אוגר מקטע** (segment) **ואוגר היסט** (Offset). לצורך חישוב הכתובת האמיתית המעבד מחבר את התוכן של שני האוגרים בחיבור הזה: המעבד לוקח את אוגר המקטע, מזיז אותו שמאלה בספרה הקסדצימלית אחת ומחבר אליו את אוגר ההיסט.

הבית התחתון (LSB) של מילה יאוחסן בכתובת ראשונה **והבית העליון** (MSB) של המילה יאוחסן בכתובת הבאה. אחסון של כתובת בזיכרון בתוך משתנה המשמש כמצביע לכתובת תתבצע באופן דומה.

סוגי זיכרון

על הלוח הראשי במחשב האישי נמצאים שני סוגי זיכרון עיקריים: זיכרון התקני ROM BIOS, וזיכרון ראשי - RAM.

זיכרון ROM מותקן בלוח הראשי של כל מחשב אישי ומאפשר קריאה בלבד. הוא מכיל את **תוכנית ההפעלה הראשונית** של המחשב האישי, הקרויה **ROM BIOS**. הוא משמש למחשב האישי לפעולת ה**אתחול** (Boot) ולהפעלת התקני חומרה וכרטיסי תיאום המחוברים אליו. זיכרון ROM אינו נמחק עם כיבוי המחשב, בניגוד לזיכרון RAM. הוא קטן יחסית לזיכרון RAM וגודלו בדרך כלל 16KB ועד 256KB במחשבים המבוססים על מעבדי 80286 ומעלה.

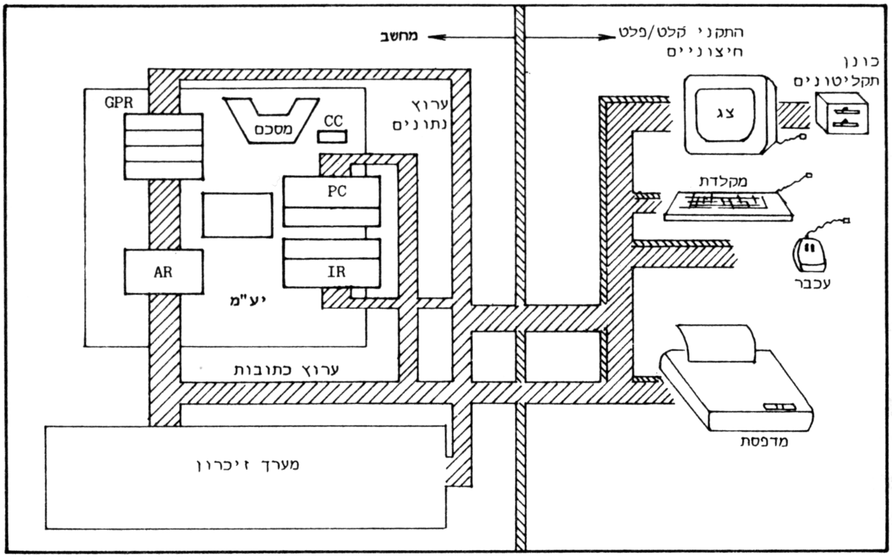
הזיכרון הראשי, הקרוי **RAM**, משמש את המחשב האישי בפעילות השוטפת. הוא מאפשר למעבד פעולות קריאה וכתיבה תוך כדי פעילות המערכת. תוכנית המתבצעת במחשב חייבת להימצא בזיכרון הראשי, כדי שניתן יהיה לבצע אותה. התקשורת בין המעבד להתקנים ההיקפיים מתבצעת בחלקה באמצעות אזורים בזיכרון הראשי. זיכרון RAM נדיף, כלומר בכיבוי המתח הוא נמחק והתוכן שלו אובד.

זיכרון RAM שעל הלוח הראשי נקרא **RAM דינמי** (DRAM), מכיון שדרוש **רענון** (Refreshing) שוטף של תוכנו כדי שלא יימחק. הפעולה נעשית בחומרה והמשתמש אינו מעורב בה. רכיבי הזיכרון נמצאים על הלוח הראשי במחשב ואם נדרש, גם על כרטיסי הרחבה המתחברים ללוח הראשי. במחשבים האישיים מותקנים רכיבי זיכרון מסוג SIMM הנראים כמו כרטיס מתאם קטן ומכילים נתונים בקיבולת של 1MB ועד 16MB.
רכיבים אלה מאפשרים גישה לזיכרון (בהתאם לסוג הרכיב) ברוחב 8, 16, 32 סיביות.
ארגון הזיכרון "שקוף" לתוכניות מערכת ההפעלה ולתוכניות היישום השונות, ולמשתמש אין אפשרות לשנות דבר. מרחב הזיכרון המירבי שניתן לגישה על ידי המעבד מחולק לשני אזורים עיקריים: מרחב התוכנה (0-640K) ומרחב החומרה (640K-1M). הזיכרון שמעל **1M** הוא זיכרון הרחבה (Extended), שבו יכולה מערכת ההפעלה להשתמש כאזור זיכרון נוסף.

אפיק הנתונים והעולם החיצון

יחידות קלט/פלט - Input/Output

בין המחשב לבין העולם החיצון קיים קשר רב-גווני באמצעות מסופים, מדפסות, עכברים, עטי אור, מקלדת ועוד. הדרך בה מקושר המחשב להתקנים אלה מורכבת למדי, ולא נדון בכך. במחשבים שונים השיטות שונות. בדרך כלל משתמשים ברכיבים אלקטרוניים נוספים כמתווכים בין המחשב לבין התקנים אלה. ברוב המחשבים מסוג מיקרו ומיני מנוצלים אפיקי הנתונים של מערך הזיכרון בתוספת קווי בקרה לצורך עבודה עם ההתקנים. הפעולות להבאת הנתונים לתוך היע"מ ולתוך הזיכרון נקראות **קלט** ופעולות הוצאת נתונים מהיע"מ ומהזיכרון נקראות **פלט**.



*תרשים א.7 : **מערך קלט/פלט***

בשרטוט ניתן להבחין בחיבור בין היע״מ לבין מערך הזיכרון וההתקנים החיצוניים באמצעות שני ערוצים עיקריים: אפיק הכתובות ואפיק הנתונים. אפיקים אלה משותפים לזיכרון ולהתקנים שסביבו.

הנתונים עוברים במחשב באמצעות **אפיק הנתונים** (Data Bus). מבנה האפיק משפיע על העברת הנתונים בין ההתקנים אל הלוח הראשי והזיכרון. אנו מכירים מספר סוגי אפיקים:

- אפיק ISA
- אפיק EISA
- אפיק VESA (Local BUS)
- אפיק PCI
- מיקרו-אפיק

גישה למרחב הקלט/פלט

מרחב הקלט/פלט (Input/Output - I/O) הוא תוספת למרחב הזיכרון במעבד. המעבד ניגש למרחב זה על ידי הוצאת כתובת אל אפיק הכתובות. המעבד מאפשר למתכנת לשלוט על מרחב זיכרון זה באמצעות שתי פקודות מיוחדות השייכות לאוסף הפקודות של המעבד: הפקודות IN ו-OUT מאפשרות להעביר נתונים מהמעבד למרחב הקלט/פלט ולהיפך. בפעולות אלו משתתף אוגר AL להעברת נתונים ברוחב 8 סיביות, או אוגר AX להעברת נתונים ברוחב 16 סיביות. אל כתובות 0 עד 00-FFh במרחב הקלט/פלט ניתן לגשת במיעון ישיר. גישה לכתובות 100h ואילך במרחב הקלט/פלט נעשית באמצעות אוגר DX שערכו מציין כתובת במרחב הקלט/פלט.

שיטות לייצוג נתונים בשפת C

המחשב האלקטרוני מיטיב לטפל בחישובים מספריים ובביצוע פעולות מתמטיות. יכולת זו נגזרת מיכולת האחסון הגבוהה שלו וממהירות פעולתו. יחידת המידע הבסיסית במחשב הינה ה**סיבית** (Bit). ערכה של סיבית בודדת יכול להיות 0 (אפס) או 1 בלבד. כדי לייצג מידע מגוון יותר מ-1 ו-0, צריך לקבץ יחדיו מספר סיביות. לדוגמה, באמצעות שתי סיביות ניתן לייצג ארבעה ערכים: 00, 01, 10 ו-11 (0, 1, 2 ו-3).

יחידות המידע במחשב מיוצגות באמצעות קבוצה של 8 סיביות הקרויות **בית** (Byte) או **תו**, או בקבוצות גדולות יותר הקרויות **מילה** (Word). גודל המילה המקובל הינו 32 סיביות. 16 סיביות הינן **חצי-מילה** (Half-Word). באמצעות הבית ניתן לייצג 256 ערכים (מ-0 עד 255), שהם מספר הצירופים שניתן להרכיב מ-8 סיביות (2^8=256). מספרים אלה קרויים **מספרים בינאריים** (Binary numbers). ניתן לקבץ יחדיו סיביות רבות יותר כדי לקבל מספרים בערכים גדולים יותר. למשל, אם נתייחס לזוג בתים, כלומר 16 סיביות - נוכל לייצג את הערך 2^16-1=65,535.

שפת C מאפשרת למתכנת לציין ערכים בשלוש שיטות ייצוג שונות: עשרונית, אוקטלית והקסדצימלית. לא קיים ייצוג ישיר של מספרים בינאריים.

ייצוג עשרוני:	כל מספר, בו מצוינות הספרות 0-9, נחשב ב-C למספר בייצוג עשרוני.
ייצוג אוקטלי:	כל מספר, המתחיל בספרה 0, ייחשב למספר בייצוג אוקטלי. שפת C מחייבת ששאר הספרות במספר תהיינה 0 עד 7. לדוגמה: 072, 025, 017.
ייצוג הקסדצימלי:	כל מספר בעל הקידומת 0x (אפס והאות הקטנה x) ייחשב למספר בייצוג הקסדצימלי. לדוגמה: 0x2d, 0x5E, 0xFA, 0x6D, 0x42.

שפת C מאפשרת למתכנת לבצע פעולות ברמה של סיביות ולטפל גם במספרים בינאריים, אך בדרך עקיפה. בנספח זה נציג תיאור של שיטות הייצוג של מספרים בינאריים ואריתמטיקה במספרים בינאריים. אנו ממליצים ללמוד חומר זה, במידה ואינך שולט בו כראוי, כדי שתוכל להפעיל טוב יותר את שפת C, בנושאים מורכבים.

ייצוג דצימלי (עשרוני)

נפתח בהסבר של הייצוג הדצימלי (עשרוני) שבו אנו משתמשים. נתבונן במספר 473 המיוצג בבסיס 10. נוכל לרשום:

$$473 = 4*100 + 7*10 + 3*1$$

או:

$$473 = 4*10^2 + 7*10^1 + 3*10^0$$

במספרים עשרוניים קיימות לרשותנו 10 ספרות (כי מספר הספרות הוא כסדר הבסיס). הספרה הראשונה הינה 0 ולכן עשר הספרות של הבסיס העשרוני הינן 0-9. כאשר מחברים שני מספרים נרשום אותם זה תחת זה באופן שטורי היחידות יהיו מכוונים בימין. נחבר כל טור בנפרד. בכל מקרה בו יתקבל סכום שתי ספרות הגדול מ-9 ייווצר **נשוא** (Carry), אשר נעביר לטור הבא, השמאלי יותר. לדוגמה, נחבר 1053 + 473:

```
נשוא  ---->  1
        1 0 5 3
          4 7 3
        -------
        1 5 2 6
```

ייצוג בינארי

מספרים בינאריים מכילים את הספרות 1 או 0 בלבד, ועל כן נאמר שהם מיוצגים בבסיס 2. נחשב את ערכו של מספר בינארי:

$$01101011 =$$

$$= 0*2^7 + 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 =$$

$$= 0 + 1*64 + 1*32 + 0 + 1*8 + 0 + 1*2 + 1*1 =$$

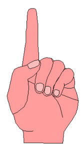
$$= 64+32+8+2+1 = 107$$

בייצוג עשרוני נקבל: $01101011_2 = 107_{10}$

נחבר שני מספרים בינאריים בדרך דומה לחיבור של שני מספרים עשרוניים. כלומר, נרשום את שני המספרים כך שהספרות המייצגות את היחידות תהיינה זו מעל זו, ונחבר כל טור בנפרד. אם התוצאה גדולה מ-1, נעביר נשוא לטור הבא. לדוגמה, נחבר 01101011 + 1011:

```
נשוא  ---->  1 1 1 1
     0 1 1 0 1 0 1 1
             1 1 0 1
     ---------------
     0 1 1 1 1 0 0 0
```

בדוק בעצמך את ערכי המחוברים ואת התוצאה, ובחן את עצמך.



שים לב!

כללי החיבור בבסיס 2:

```
0 + 0 = 0        (0)
0 + 1 = 1        (1)
1 + 1 = 10       (2)
1 + 1 + 1 = 11   (3)
```

(שורה (2) מודגשת)

הפיכת ייצוג עשרוני לייצוג בינארי

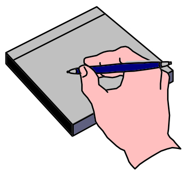
בכדי לקבל ייצוג בינארי של מספר עשרוני, עליך לבצע את התהליך הבא: חלק את המספר העשרוני ב-2 ורשום את המנה ואת השארית. כעת חלק את המנה ב-2 ורשום את המנה החדשה והשארית החדשה. המשך בתהליך זה עד לקבלת מנה "אפס". המספר בייצוג הבינארי הוא סדרת השאריות, בסדר כתיבה הפוך. לדוגמה, נבצע את התהליך עבור המספר העשרוני 107:

מנה	שארית
107 / 2 = 53	1
53 / 2 = 26	1
26 / 2 = 13	0
13 / 2 = 6	1
6 / 2 = 3	0
3 / 2 = 1	1
1 / 2 = 0	1
0	0

סדר קריאת המספר בייצוג הבינארי = => 0 1 1 0 1 0 1 1

ייצוג מספרים בינאריים שליליים

כאשר עוסקים במספרים עשרוניים, אנו מציינים מספר שלילי על ידי הוספת סימן מינוס ("-") משמאל למספר. כדי לחבר מספר שלילי עשרוני למספר עשרוני חיובי, אנו נוקטים בתהליך חיסור במקום בתהליך חיבור. הסימן "-" הוא סימן גרפי המורה לנו כיצד לנהוג.



הערה!

במקרה של מספרים בינאריים וייצוגם במחשב באמצעות סיביות, משתמשים בסיבית השמאלית ביותר לסימן. אם היא בעלת ערך 0 - המספר חיובי, ואם 1 - המספר שלילי. כלל זה גורם לכך שטווח המספרים שנוכל לייצג על ידי בית או מילה, יהיה קטן ממספר הסיביות: טווח הערכים שאפשר לייצג בבית הוא (2^7) -128 עד +127.

כדי לחבר שני מספרים בייצוג בינארי עלינו לבדוק תחילה את ערך סיבית הסימן, ובהתאם לכך לחבר או לחסר את המספרים. אולם, כדי שאפשר יהיה לחבר מספרים בינאריים חיוביים ושליליים באמצעות אותו התהליך, נשתמש בשיטת ייצוג שונה למספרים שליליים. שיטת ייצוג זו הינה **שיטת משלים-2** (Two's complement), אשר נתאר אותה בקצרה.

שיטת משלים-2

מספר בינארי חיובי הוא מספר שהסיבית השמאלית שלו היא 0. לדוגמה, במספר בינארי חיובי בעל 8 סיביות, שבע הסיביות הימניות מייצגות את הערך והסיבית השמאלית - את הסימן 0. בכדי לקבל ייצוג של מספר בינארי שלילי בשיטת **משלים-2**

(Two's Complement) נרשום אותו תחילה כמספר חיובי, נחליף כל ספרה 1 ב-0 וכל ספרה 0 ב-1 ונוסיף לתוצאה 1.

דוגמה:

```
המספר הנתון (49)      0 0 1 1 0 0 0 1
החלפת 1 ו-0           1 1 0 0 1 1 1 0
הוספת 1                           + 1
                      ---------------
                      1 1 0 0 1 1 1 1
```

התקבל המספר הבינארי שערכו 49- בשיטת משלים-2. נחבר מספר זה למספר 83 (בייצוגו הבינארי) ונקבל:

$11001111_2 = -49_{10}$
$01010011_2 = \ 83_{10}$

```
 נשוא  ----> 1 1 1 1
       0 1 0 1 0 0 1 1
       1 1 0 0 1 1 1 1
       ---------------
       0 0 1 0 0 0 1 0 = 1*2^5 + 1*2^1 = 32 + 2 = 34
```

ואכן, $83 + (-49) = 83 - 49 = 34$

לסיכום, כדי לחסר מספר מחברו, הופכים אותו למספר שלילי בשיטת משלים-2 ומחברים אותו. שים לב, שבמספר בינארי שלילי בייצוג משלים-2 הסיבית השמאלית ביותר הינה 1.

כאשר החיבור גורם לכך שהתוצאה תהיה מחוץ לטווח הייצוג האפשרי לפי מספר הסיביות מקבלים מצב שגיאה, או אובדן מידע. לדוגמה, טווח ייצוג באמצעות בית בודד הוא 128- עד 127+ וטווח הייצוג באמצעות שני בתים הוא 32,768- עד 32,767+. מצב שגוי זה נקרא **גלישה** (Overflow).

תכונות של מספרים בינאריים

לסיכום, נציג רשימת תכונות של מספרים בינאריים.

א. במספר בינארי כלשהו, הסיבית הימנית ביותר קובעת אם הוא זוגי או אי-זוגי:

- 0 מספר זוגי.
- 1 מספר אי-זוגי.

ב. במספר בינארי המיוצג בשיטת משלים-2 הסיבית השמאלית ביותר קובעת את הסימן:

- 0 מספר חיובי.
- 1 מספר שלילי.

ג. אם נסיט את כל הסיביות של מספר בינארי מקום אחד שמאלה ונוסיף סיבית 0 מימין, יהיה ערכו של המספר הבינארי המתקבל גדול פי 2 מהמספר המקורי (דומה לכפל ב-10 בשיטה העשרונית).

ד. אם נסיט את כל הסיביות של מספר בינארי מקום אחד ימינה ונמחק את הסיבית הימנית ביותר, תהיה זו חלוקה של המספר המקורי ב-2.

ה. את הערך 1- נציג כך בשיטת משלים 2 :

ייצוג בינארי	11111111_2
ייצוג הקסדצימלי	FF_{16}

ו. המספר הבינארי בן 8 סיביות : 10000000_2 הינו בעל מעמד מיוחד. ייצוגו בשיטת משלים-2 תיתן את אותו מספר עצמו. מסיבה זו, במערכות מחשב מסוימות מוגדר צירוף זה כבלתי חוקי, ובמערכות אחרות מוגדר הצירוף כבעל הערך 128- וכך מורחב התחום המותר עד 128-. בדומה, נקבל עבור מספר בינארי בן 16 סיביות את הטווח עד 32,768-.

ייצוג הקסדצימלי ואוקטלי

בכדי לקבל ייצוג הקסדצימלי של מספר עשרוני עליך לבצע את התהליך הבא : חלק את המספר העשרוני ב-16 ורשום את המנה ואת השארית. חלק את המנה ב- 16 ורשום את המנה החדשה ואת השארית החדשה. המשך בתהליך זה עד לקבלת מנה שקטנה בערכה מ- 16. המספר בייצוג הקסדצימלי הוא סדרת השאריות, בסדר כתיבה הפוך.

לדוגמה, נבצע את התהליך עבור המספר העשרוני 1234 :

↑ שארית		מנה ↓	
2	77	=	1234 / 16
13	4	=	77 / 16
4	0	=	4 / 16

סדר קריאת המספרים בייצוג הקסדצימלי : $(4D2)_{16}$. אך יש שיטה פשוטה יותר. קריאה של מספרים בינאריים הינה מסורבלת בגלל מספר הספרות הרב. נוח יותר להשתמש בשיטת ייצוג הקסדצימלית או אוקטלית, כדי לייצג מספרים בינאריים, בגלל התאמה בבסיס. בשיטה ההקסדצימלית, לפי בסיס 16 (2^4), טווח המספרים הוא 0 עד 15. בשיטה זו נשתמש במספרים 0 עד 9 ו-A עד F, עבור 16 הספרות. בשיטה האוקטלית, לפי בסיס 8 (2^3), טווח המספרים הוא 0 עד 7.

כאשר רוצים לייצג סידרה של ספרות בינאריות על ידי מספר אוקטלי או הקסדצימלי, יש לחלק את הספרות הבינאריות לקבוצות של 3 ספרות או 4 ספרות בהתאמה. אם מספר הספרות אינו מתאים, יש להוסיף אפסים משמאל. אם המספר הוא שלילי בשיטת משלים-2 יש להוסיף ספרות 1. הוספת ספרות משמאל נקראת **הרחבת הסימן** (Sign Extend) ראה טבלה א.1.

טבלה א.1 : ***הצגת מספרים ב-4 בסיסים שונים***

מספר אוקטלי	**מספר הקסדצימלי**	**מספר בינארי**	**ערך עשרוני**
0	0	0 0 0 0 0	0
1	1	0 0 0 0 1	1
2	2	0 0 0 1 0	2
3	3	0 0 0 1 1	3
4	4	0 0 1 0 0	4
5	5	0 0 1 0 1	5
6	6	0 0 1 1 0	6
7	7	0 0 1 1 1	7
10	8	0 1 0 0 0	8
11	9	0 1 0 0 1	9
12	A (*)	0 1 0 1 0	10
13	B	0 1 0 1 1	11
14	C	0 1 1 0 0	12
15	D	0 1 1 0 1	13
16	E	0 1 1 1 0	14
17	F	0 1 1 1 1	15
20	10	1 0 0 0 0	16

(*) אפשר לכתוב את האותיות A עד F כאותיות "קטנות": a, b,... f.

נבחן עתה את הערכים העשרוניים של מספרים בבסיס אוקטלי והקסדצימלי.

לדוגמה, נהפוך את המספר הבינארי הבא לייצוג אוקטלי: **01011010**. נחלק את המספר (החל מימין) לקבוצות בנות 3 סיביות:

001	011	010
↓	↓	↓
1	3	2

נקבל את המספר האוקטלי:

כלומר, $01011010_2 = 132_8$

לדוגמה, נתון מספר בינארי בן 8 סיביות: **01101101**. נפצל אותו לשתי קבוצות בנות 4 סיביות כל אחת ונקבל:

ייצוג בינארי:	1101	0110
ייצוג הקסדצימלי:	D	6

ניתן בדרך דומה להציג מספר הקסדצימלי כמספר בינארי: מחליפים כל ספרה ב-4 סיביות מתאימות. ערכו העשרוני של המספר ההקסדצימלי הזה הוא:

$$6D_{16} = 6*16^1 + D*16^0 = 96 + 13 = 109_{10}$$

חיבור מספרים

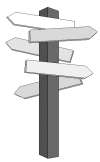
אפשר לחבר מספרים בבסיס הקסדצימלי או אוקטלי כמו שמחברים מספרים בכל בסיס אחר. כאשר מחברים מספרים הקסדצימלים הנשוא מחושב עבור תוצאות סיכום הגדולות מ-F. בחיבור של מספרים אוקטליים, הנשוא מחושב עבור תוצאת סיכום הגדולות מ-7.

טבלה א.2 : ***סיכום כללי ההמרה מבסיס לבסיס***

מספר מבסיס x	למספר בבסיס y	סדר הפעולות
10	x	חלק את המספר בבסיס 10 בבסיס היעד X, עד אשר החלק השלם שווה ל- 0. התוצאה המתקבלת היא סדרת השאריות בסדר כתיבה הפוך.
x	10	כפול את הספרות במספר X בחזקות עולות של X מהסוף להתחלה, החל בחזקה 0.
x	y	העבר תחילה את המספר בבסיס X למספר בבסיס 10, ולאחר מכן העבר אותו למספר בבסיס Y.
2	8	חלק את המספר בבסיס 2 לקבוצות של 4 החל מימין.
2	16	חלק את המספר בבסיס 2 לקבוצות של 4 החל מימין.
8 16 10 16	2 2 16 10	העבר את המספר בבסיס X למספר בבסיס 10, ולאחר מכן העבר אותו למספר בבסיס Y.

קוד ייצוג תווים

המחשב מקבל מהמשתמש את הקלט מהמקלדת ומאמצעי קלט חיצוניים, כמו סורק, למשל, ומוציא פלט מוקרן על מסך, או מודפס על נייר במדפסת. המחשב בעצמו מתרגם את התווים, מספרים ותמונות לשפה שהוא "מבין" (פעולה זו מאיטה את פעולות המחשב). הנתונים מיוצגים ברכיבי המחשב בשיטה הבינארית הפועלת לפי בסיס 2, שבה קיימים רק שני צירופים שונים (0, 1). בעזרת צירופים אלה יוצרים את התבניות של כל התווים האפשריים. כלומר, סדרות של מספרים בינאריים.



נקודות ציון!

- *כל ההתקנים של המחשב פועלים בשיטת הייצוג הבינארית.*
- *מטעמי נוחות אנו מציגים את סדרת הסיביות בחלוקה ל-4, בשיטה ההקסדצימלית.*

תקן ASCII
American Standard Code for Information Interchange

תקן ASCII או **קוד ASCII**, הוא תקן בינלאומי לייצוג תווים, כגון: אותיות, ספרות, תווים גרפיים על בסיסי טקסט, תווים בקרה מיוחדים. תקן זה נפוץ בעיקר במיקרו מחשבים. על פי שיטה זו כל תו מיוצג על ידי 8 סיביות (בית אחד) ומזוהה באופן חד ערכי. בעזרת שיטת ייצוג זו ניתן לייצג 256 תווים שונים (המסומנים בערכים הדצימליים 0 עד 255), או 2^8. כאשר נדליק את כל הסיביות בבית אחד נקבל:

$$(1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)_2 = (255)_{10}$$

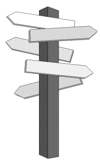
לכן, המספר הגבוה ביותר שיכול להכיל הטיפוס unsigned char הוא 255. יחד עם הערך 0, לפנינו 256 אפשרויות שונות. האם זכור לך מדוע בית אחד יכול להכיל מספר בעל סימן בטווח 127- עד 127+ בלבד?

אם כך, כיצד נייצג את התו "A"

נבדוק בטבלת ASCII המצורפת, וניווכח שערכו המספרי של התו "A" הוא 65 בבסיס 10. נמיר את הערך הזה מבסיס 10 עשרוני לבסיס 2 בינארי, ונקבל:

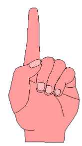
0	1	0	0	0	0	0	1	
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	64	0	0	0	0	0	1	
	64			+			1	$= (65)_{10}$

מכאן, שהתו A מיוצג בבית כך: $(0\ 1\ 0\ 0\ 0\ 0\ 0\ 1)_2$.



נקודת ציון!

משפטים (מחרוזות), נתונים שונים, ותוכניות מיוצגות על ידי סדרות ארוכות מאוד של ספרות 0, 1.



שים לב!

טבלת ASCII מכילה תווים שונים בסדר הבא: תווי בקרה, תווים מיוחדים, ספרות, אותיות אנגליות גדולות, אותיות אנגליות קטנות, אותיות עבריות, תווי גרפיקה על בסיס טקסט.

תקן EBCDIC
Extended Binary Coded Decimal Interchange Code

תקן EBCDIC הוא תקן בינלאומי לייצוג תווים כגון: אותיות, ספרות, תווי בקרה מיוחדים. התקן נפוץ בעיקר במחשבי IBM גדולים. על פי שיטה זו, בדומה לתקן ASCII, כל תו מיוצג בעזרת 8 סיביות, ולכן ניתן לייצג 2^8 תווים שונים, כלומר 256 תווים (0-255).

טבלת קוד ASCII

Decimal	Hex	Octal	Binary	ASCII
0	00	000	00000000	NUL (Null)
1	01	001	00000001	SOH (Start of header)
2	02	002	00000010	STX (Start of text)
3	03	003	00000011	ETX (End of text)
4	04	004	00000100	EOT (End of trans)
5	05	005	00000101	ENQ (Enquiry)
6	06	006	00000110	ACK (Positive acknowledge)
7	07	007	00000111	BEL (Bell)
8	08	010	00001000	BS (Back space)
9	09	011	00001001	HT (Horizontal tab)
10	0A	012	00001010	LF (Line feed)
11	0B	013	00001011	VT (Vertical tab)
12	0C	014	00001100	FF (From feed)
13	0D	015	00001101	CR (Carriage return)
14	0E	016	00001110	SO (Shift out)
15	0F	017	00001111	SI (Shift in)
16	10	020	00010000	DLE (Data link escape)
17	11	021	00010001	DC1 (Device control 1)
18	12	022	00010010	DC2 (Device control 2)
19	13	023	00010011	DC3 (Device control 3)
20	14	025	00010100	DC4 (Device control 4)
21	15	024	00010101	NAK (Negative acknowledge)
22	16	026	00010110	SYN (Synchronous idle)
23	17	027	00010111	ETB (End of trans block)
24	18	030	00011000	CAN (Cancel)
25	19	031	00011001	EM (End of Medium)
26	1A	032	00011010	SUB (Substiute)
27	1B	033	00011011	ESC (Escape)
28	1C	034	00011100	FS (File separator)
29	1D	035	00011101	GS (Group separator)
30	1E	036	00011110	RS (Record separator)
31	1F	037	00011111	US (Unit separator)
32	20	040	00100000	SP (Space)

Decimal	Hex	Octal	Binary	ASCII
33	21	041	00100001	!
34	22	042	00100010	"
35	23	043	00100011	#
36	24	044	00100100	$
37	25	045	00100101	%
38	26	046	00100110	&
39	27	047	00100111	'
40	28	050	00101000	(
41	29	051	00101001	)
42	2A	052	00101010	*
43	2B	053	00101011	+
44	2C	054	00101100	,
45	2D	055	00101101	-
46	2E	056	00101110	.
47	2F	057	00101111	/
48	30	060	00110000	0
49	31	061	00110001	1
50	32	062	00110010	2
51	33	063	00110011	3
52	34	064	00110100	4
53	35	065	00110101	5
54	36	066	00110110	6
55	37	067	00110111	7
56	38	070	00111000	8
57	39	071	00111001	9
58	3A	072	00111010	:
59	3B	073	00111011	;
60	3C	074	00111100	<
61	3D	075	00111101	=
62	3E	076	00111110	>
63	3F	077	00111111	?
64	40	100	01000000	@
65	41	101	01000001	A
66	42	102	01000010	B
67	43	103	01000011	C
68	44	104	01000100	D
69	45	105	01000101	E
70	46	106	01000110	F
71	47	107	01000111	G
72	48	110	01001000	H
73	49	111	01001001	I
74	4A	112	01001010	J
75	4B	113	01001011	K
76	4C	114	01001100	L

Decimal	Hex	Octal	Binary	ASCII
77	4D	115	01001101	M
78	4E	116	01001110	N
79	4F	117	01001111	O
80	50	120	01010000	P
81	51	121	01010001	Q
82	52	122	01010010	R
83	53	123	01010011	S
84	54	124	01010100	T
85	55	125	01010101	U
86	56	126	01010110	V
87	57	127	01010011	W
88	58	130	01011000	X
89	59	131	01011001	Y
90	5A	132	01011010	Z
91	5B	133	01011011	[
92	5C	134	01011100	\
93	5D	135	01011101	]
94	5E	136	01011110	^
95	5F	137	01011111	
96	60	140	01100000	`
97	61	141	01100001	a
98	62	142	01100010	b
99	63	143	01100011	c
100	64	144	01100100	d
101	65	145	01100101	e
102	66	146	01100110	f
103	67	147	01100111	g
104	68	150	01101000	h
105	69	151	01101001	i
106	6A	152	01101010	j
107	6B	153	01101011	k
108	6C	154	01101100	l
109	6D	155	01101101	m
110	6E	156	01101110	n
111	6F	157	01101111	o
112	70	160	01110000	p
113	71	161	01110001	q
114	72	162	01110010	r
115	73	163	01110011	s
116	74	164	01110100	t
117	75	165	01110101	u
118	76	166	01110110	v
119	77	167	01110111	w
120	78	170	01111000	x

Decimal	Hex	Octal	Binary	ASCII
121	79	171	01111001	y
122	7A	172	01111010	z
123	7B	173	01111011	{
124	7C	174	01111100	¦
125	7D	175	01111101	}
126	7E	176	01111110	~
127	7F	177	01111111	
128	80	200	10000000	א
129	81	201	10000001	ב
130	82	202	10000010	ג
131	83	203	10000011	ד
132	84	204	10000100	ה
133	85	205	10000101	ו
134	86	206	10000110	ז
135	87	207	10000111	ח
136	88	210	10001000	ט
137	89	211	10001001	י
138	8A	212	10001010	ך
139	8B	213	10001011	כ
140	8C	214	10001100	ל
141	8D	215	10001101	ם
142	8E	216	10001110	מ
143	8F	217	10001111	ן
144	90	220	10010000	נ
145	91	221	10010001	ס
146	92	222	10010010	ע
147	93	223	10010011	ף
148	94	224	10010100	פ
149	95	225	10010101	ץ
150	96	226	10010110	צ
151	97	227	10010111	ק
152	98	230	10011000	ר
153	99	231	10011001	ש
154	9A	232	10011010	ת
155	9B	233	10011011	ø
156	9C	234	10011100	£
157	9D	235	10011101	Ø
158	9E	236	10011110	Ŀ
159	9F	237	10011111	ŀ
160	A0	240	10100000	á
161	A1	241	10100001	í
162	A2	242	10100010	ó
163	A3	243	10100011	ú
164	A4	244	10100100	ñ
165	A5	245	10100101	Ñ

Decimal	Hex	Octal	Binary	ASCII
166	A6	246	10100110	õ
167	A7	247	10100111	Õ
168	A8	250	10101000	¿
169	A9	251	10101001	ã
170	AA	252	10101010	Ã
171	AB	253	10101011	ℓ
172	AC	254	10101100	ħ
173	AD	255	10101101	¡
174	AE	256	10101110	з
175	AF	257	10101111	¤
176	B0	260	10110000	░
177	B1	261	10110001	▒
178	B2	262	10110010	▓
179	B3	263	10110011	░
180	B4	264	10110100	┤
181	B5	265	10110101	╡
182	B6	266	10110110	╢
183	B7	267	10110111	╖
184	B8	270	10111000	╕
185	B9	271	10111001	╣
186	BA	272	10111010	║
187	BB	273	10111011	╗
188	BC	274	10111100	╝
189	BD	275	10111101	╜
190	BE	276	10111110	╛
191	BF	277	10111111	┐
192	C0	300	11100000	└
193	C1	301	11000001	┴
194	C2	302	11000010	┬
195	C3	303	11000011	├
196	C4	304	11000100	─
197	C5	305	11000101	┼
198	C6	306	11000110	╞
199	C7	307	11000111	╟
200	C8	310	11001000	╚
201	C9	311	11001001	╔
202	CA	312	11001010	╩
203	CB	313	11001011	╦
204	CC	314	11001100	╠
205	CD	315	11001101	═
206	CE	316	11001110	╬
207	CF	317	11001111	╧
208	D0	320	11010000	╨
209	D1	321	11010001	╤

Decimal	Hex	Octal	Binary	ASCII
210	D2	322	11010010	╥
211	D3	323	11010011	╙
212	D4	324	11010100	╘
213	D5	325	11010101	╒
214	D6	326	11010110	╓
215	D7	327	11010111	╫
216	D8	330	11011000	╪
217	D9	331	11011001	┘
218	DA	332	11011010	┌
219	DB	333	11011011	█
220	DC	334	11011100	▄
221	DD	335	11011101	▌
222	DE	336	11011110	▐
223	DF	337	11011111	▀
224	E0	340	11100000	α
225	E1	341	11100001	β
226	E2	342	11100010	Γ
227	E3	343	11100011	π
228	E4	344	11100100	Σ
229	E5	345	11100101	σ
230	E6	346	11100110	µ
231	E7	347	11100111	τ
232	E8	350	11101000	Φ
233	E9	351	11101001	θ
234	EA	352	11101010	Ω
235	EB	353	11101011	δ
236	EC	354	11101100	∞
237	ED	355	11101101	φ
238	EE	356	11101110	∈
239	EF	357	11101111	∩
240	F0	360	11110000	≡
241	F1	361	11110001	±
242	F2	362	11110010	≥
243	F3	363	11110011	≤
244	F4	364	11110100	⌠
245	F5	365	11110101	⌡
246	F6	366	11110110	÷
247	F7	367	11110111	≈
248	F8	370	11111000	°
249	F9	371	11111001	•
250	FA	372	11111010	·
251	FB	373	11111011	√
252	FC	374	11111100	^{n}
253	FD	375	11111101	^{2}
254	FE	376	11111110	■
255	FF	377	11111111	

נספח ב'

נהלים לכתיבת תוכנית ב-C

באדיבות **סיון מחשבים**

נספח זה מתאר נוהל לכתיבה תקנית בשפות C/C++, ומבוסס על חומר אשר מקובל ב**מכללת סיון מחשבים**. הכללים המתוארים כאן הם קווים מנחים כלליים, והשימוש בהם כפוף לשכל הישר של המתכנת בשפת C או C++.

מטרה

הרגלי כתיבה נכונים צריכים לתרום להשגת המטרות הבאות:

- תוכניות קלות לקריאה, הבנה, ותחזוקה.
- הקטנת כמות השגיאות (bugs).
- פישוט תהליך הניפוי (debugging).
- כתיבה ממושמעת ומבוקרת, על פי כללים קבועים.
- מניעת מגבלות מיותרות להרחבת התוכנית בעתיד.
- יתר יעילות בהידור התוכנית.
- כתיבת קוד יביל (portable).
- מודולריות ותמיכה בשימוש חוזר בקוד.

גופן הכתיבה והבחנה בין סוגי אותיות

העבודה על המסך נעשית ב**עורך** (editor) מסוג ASCII, שבו הגופן מייצר תווים אחידים בגודלם (לעומת תווים יחסיים כמקובל בטקסט).

לשפת C יש מהדרים רבים. יש מהדרים שבהם יש הבחנה בין אותיות גדולות וקטנות (באנגלית) ולכן, abc **אינו שווה** ל-ABC. לעומתם, יש מהדרים שאינם עושים את ההבחנה הזו ועבורם abc **שווה** ל-ABC. אנו משתמשים באותיות גדולות וקטנות לשם נוחות קריאת הקוד, השמות וכד'. עם זאת, ההמלצה היא להימנע מהתייחסות למשתנים כשונים, בגלל שוני בסוג האות שלהם.

סוגי קבצים ושמות קבצים

קובצי קוד (source)

קובץ קוד C יקבל סיומת "C" ושם המתאר את תפקידו. אין הבחנה בין אותיות קטנות וגדולות. על כן, אפשר לכתוב Budget.c, או BUDGET.C, למשל. על כן, שני השמות האלה **זהים**, למרות שהם כתובים באותיות שונות.

קובצי כותרת (header) - קובצי include פרטיים

קובץ כותרת פרטי, המשמש קובץ קוד מסוים בלבד, יקבל את הסיומת "H", ושם כשם קובץ הקוד שלו, בתוספת **קו תחתון** (underline). כך למשל, יש לכתוב INV_.H. אולם, אם שם קובץ הקוד כבר כולל 8 תווים, יוחלף התו האחרון בקו תחתי.

קובצי כותרת (header) - קובצי include לשימוש הכלל

קובץ כותרת שנועד לאפשר לקובצי קוד אחרים שימוש בקובץ קוד מסוים (מכיל הכרזות לפונקציות גלובליות וכד') יקבל סיומת "H", ושם כשם קובץ הקוד שלו. למשל: UTILS.H.

שמות מזהים (identifiers)

שמות משתנים מכל סוג חייבים להיות שמות משמעותיים, המתארים את המטרה לה משמש המשתנה, הפונקציה, הקובץ וכו'.

משתנים מקומיים

שם משתנה מקומי (בתוך פונקציה) יורכב מאותיות קטנות, ספרות, ותו **קו-תחתון** (underline) להפרדה בין מילים. למשל, input_file_name. יש מקרים שבהם מקובל לתת שם קצר למשתנה מקומי.

- משתנה המשמש כאינדקס פשוט ללולאה, יכול להיקרא i, j, k וכדומה.
- כשפונקציה מחזירה ערך המציין הצלחה או כישלון, ניתן לשים את הערך בתוך משתנה בשם **V** (return value), בתנאי שהשימוש במשתנה מוגבל לאזור קטן בתוכנית.

משתנים גלובליים

שם משתנה גלובלי יורכב מ**אותיות מעורבות** (אותיות קטנות וגדולות), כאשר אות גדולה מציינת תחילת כל מילה. למשל, CmdLineCode. ניתן להשתמש בקו-תחתון להדגשת ההפרדה בין המשמעות הכללית של המשתנה, למשמעות המסוימת שלו. למשל, Mode_AllUsers.

מאקרו - הגדרות #define

שם מאקרו יורכב מאותיות גדולות, ספרות, קו-תחתון להפרדה בין מילים. למשל, CMD_LINE_MAINTENANCE.

פונקציות מקומיות - static

שם פונקציה המיועדת לשימוש פנימי של קובץ קוד יחיד יינתן לפי אותם כללים של משתנה מקומי. למשל, read_employee_file().

פונקציות גלובליות

שם פונקציה המיועדת לשימוש גם מחוץ לקובץ הקוד בו היא מוגדרת, יינתן לפי הכללים לכתיבת שם של משתנה גלובלי. למשל, FreeAnyLinkedList().

איברים ב-struct או union

שמות איברים במבנים יינתנו לפי הכללים לכתיבת שם של משתנה גלובלי. למשל, department.DeptNum.

שים לב לנקודה המפרידה בין שם המבנה (משמאל) לבין שם השדה שנמצא בו (מימין).

איברים ב-enum

שמות איברים באנומרטור יינתנו לפי כללי הכתיבה של שם מאקרו - אותיות גדולות ומילים מופרדות בקו-תחתון.

הזחת הקוד

הקוד נכתב משמאל לימין, וכל ההזחות מתייחסות לשוליים השמאליים של המסך.

פקודה התלויה בפקודת תנאי, או לולאה, תוסט 3 רווחים ימינה.	<pre>if (rc !=RV_OK) return(rv);</pre>
אם בלוק של קוד תלוי בפקודת תנאי או לולאה, הסוגריים המסולסלים (פתיחה וסגירה) יוסטו 3 רווחים ימינה, והפקודות בבלוק יוסטו 3 רווחים נוספים ימינה.	<pre>for (i=0; i<array_len; i++) { sigma +=array[i]; sigma_sqr +=array[i]*array[i] }</pre>

בפקודה Switch, הסוגריים המסולסלים (פתיחה וסגירה) יוסטו 3 רווחים ימינה, שורות הפקודה case יוסטו 3 רווחים נוספים ימינה, הפקודות עצמן יוסטו 3 רווחים נוספים ימינה.	

```
switch (operator)
  {
    case '+':
      rv = PerformAdd(&result, value);
      break
    case '*':
      rv = PerformMul(&result, value);
      break;
    default:
      rv = RV_BAD_OPERATOR;
  }
```

בפקודה switch הפקודות לביצוע קצרות מאוד, ולכן מותר להציב אותן בהמשך ל-case. זהו גם המקרה היחיד בו מותר לכתוב שתי פקודות בשורה אחת (פקודה כלשהי ולאחריה הפקודה break).

```
switch (operator)
  {
   case '+': result += value; break;
   case '*': result += value; break;
   case '-': result -= value; break;

   default:
    return (RV_BAD_OPERATOR);
  }
```

בהגדרת פונקציה **אין** צורך להסיט את הסוגריים המסולסלים התוחמים את הקוד.

```
void ThreeCheers()
{
  int i;


  ...
}
```

כשיש קינון של פקודות, כל פקודת **בקרה** התלויה בקודמת תוכנס לבלוק התחום בסוגריים מסולסלים, גם אם זוהי פקודה בודדת.

```
if (operator =='/')
  {
     if (value !=0)
       result /= value;
  }
```

כשיש שימוש במבנה if-else-if, והפקודה if השנייה שונה לחלוטין מהראשונה, הקינון יהיה רגיל. הפקודה else תוסט כמו הפקודה if שקדמה לה.

```
if (operator !='/')
  PerformNormalOperation(operation);
else
  {
     if (value !=0)
       result /= value;
  }
```

כשיש שימוש במבנה if-else-if, והפקודה if השנייה דומה לראשונה, הקינון יכול להיראות "מקבילי".

```
if  (stremp(operation , "add") == 0)
  result += value;
else if (stremp(operation , "mul") == 0)
  result += value;
else if (stremp(operatiom, "div") == 0)
  {
     if (value != 0)
       result /= value;
   }
```

בהגדרת משתנים ייווצרו שתי עמודות: עמודת type ועמודת שמות. תווי נקודה-פסיק (;) ייכתבו בעמודה נוספת, כדי שלא יפריעו לקריאת השמות.

```
void main(void)
{
  struct dept depatment;
  float  result;
  int    i     ;
  int    rv    ;
```

בהגדרת פונקציה, יירשמו הפרמטרים בדומה להגדרת משתנים.

```
float Calculate (float parm0 ,
        float parm1,
        char op_code )
{
```

ריווח בין פקודות

- ❖ שורת הגדרת משתנה תגדיר משתנה אחד בלבד, אלא אם היא מגדירה מספר משתנים פשוטים המשמשים לאותה מטרה (למשל, אינדקס פשוט ללולאה).

```
int i, j, k;
```

- ❖ בין שורות של הגדרת משתנים לא יהיה רווח.
- ❖ בין הגדרת המשתנים לפקודות הביצוע יהיו שתי שורות רווח (אין זו חובה, כמובן ואפשר להסתפק ברווח של שורה אחת).

❖ שורת קוד תכיל פקודה אחת בלבד (חוץ ממקרים מסוימים ב-switch).

❖ בין שתי פקודות שאינן חלק מאותה הפעולה תהיה שורת רווח אחת.

❖ מותר להצמיד שורות קוד אם הן חלק מפעולה או מטלה אחת.

```
strepy(full_name, first_name);
streat(full_name , ""   );
streat(full_name , sur_name  );
```

❖ פקודה התלויה בפקודת בקרה תיכתב בשורה שאחרי פקודת הבקרה.

```
if (value != 0)
  result /= value;
```

❖ ההפרדה הוויזואלית בין פונקציות מתקבלת על ידי תיעוד הפונקציה, כמתואר בהמשך.

פקודות בתוכנית

continue ,goto

❖ בפקודות אלו רצוי להשתמש במקרים נדירים ומיוחדים בלבד. כזכור, אנו רוצים בתכנות מבני, שבו אין מקום לפקודות אלו. רצוי לתת הסבר קצר, לצורך העתיד.

break

❖ מותר לשימוש בתוך switch (ראה להלן).

❖ אין להשתמש ב-break מחוץ ל-switch, אלא במקרים מיוחדים. רצוי לצרף הסבר קצר, שמא נרצה להבין מדוע עשינו זאת.

switch

❖ בסוף כל case שאינו אמור להמשיך ל-case הבא, יש לכתוב את הפקודה break; גם בסוף ה-case האחרון.

❖ בכל switch צריכה להיות שורת default.

for

❖ רצוי להימנע מריבוי הוראות אתחול או קידום, חוץ מבמקרים פשוטים ביותר.

לולאות for ,while ,do..while

❖ רצוי להימנע משימוש בלולאות ללא גוף. במקום while (str[i++] != '\()'); יש לכתוב:

```
while (str[i] != '\()')
  i++;
```

קינון פקודות בקרה - if ,else ,for ,while ,do..while

- ❖ רצוי להימנע מקינון של יותר מ-3 רמות.

הגדרת משתנים

- ❖ רצוי להשתמש במשתנים מקומיים-אוטומטיים בלבד, ולא במשתנים מסוג static או global (אם משתמשים במשתנים מסוגים אלה, רצוי לצרף הסבר קצר, למען הבנה טובה יותר בעתיד).
- ❖ ניתן להשתמש במשתנה גלובלי או סטטי, אם המשתנה מוגדר כקבוע (const).
- ❖ כשיש צורך ברור במשתנה שאינו מקומי, עדיף להשתמש בסוג static, כדי להגביל את **טווח ההכרה** (scope) של המשתנה לקובץ יחיד.

הגדרת קבועים

- ❖ קבועים מספריים (למשל גודל מערך) יוגדרו בעזרת define#.
- ❖ קבוצות של קבועים (למשל קודי שגיאה של פונקציה) יוגדרו בעזרת enum.
- ❖ קבוע תו - המתייחס לקוד ASCII - יוגדר כקבוע תווי, ולא מספרי. למשל, "A".
- ❖ קבועים שאמורים להיות בעלי סוג long ירשמו עם הסיומת L, למשל: 50L.
- ❖ קבועים שאמורים להיות בעלי סוג unsigned ירשמו עם סיומת U, למשל: 50000U.
- ❖ קבועים שלמים שאמורים להיות בעלי סוג float ירשמו עם סיומת 0., למשל: 50.0 (כלומר נקודה ואפס).
- ❖ קבועים בהם יש משמעות לערכים של סיביות בודדות יינתנו בבסיס 16, כשמספר הספרות תלוי בסוג הקבוע (2 ספרות ל-char, 4 ספרות ל-short, 8 ספרות ל-long). הקידומת תהיה 0x, ולא 0X (רצוי x קטנה). הספרות A-F תהיינה "גדולות". למשל,

```
#define ATTR_LIGHT_GREEN 0x0A
```

- ❖ אין להניח דבר לגבי הערך המספרי של קבועים - מלבד משתנים בוליאניים, המקבלים ערכי FALSE/TRUE. למשל, במקרה של בדיקת הקבוע NULL, במקום if(text_ptr) יש לכתוב:

```
if (text_ptr != NULL)
```

הגדרות מאקרו - #define

- יש להשתמש במאקרו בכל מקום בו השימוש ישפר את הקריאות ואת התחזוקה של התוכנית.
- מאקרו צריך להכיל ביטוי בודד בלבד:
  - מותר להשתמש בביטוי מותנה בתוך מאקרו.
  - אין לקרוא לפונקציות מתוך מאקרו.
  - אין לכלול פקודת השמה (assignment statement) בתוך מאקרו.
  - אין לכלול פקודת מורכבות (כמו if או while) בתוך מאקרו.
  - אין לכלול יותר מפקודה אחת במאקרו.
- כשערך המאקרו איננו קבוע פשוט (למשל: מספר), יש להקיף את הערך בסוגריים (שם של מאקרו אחר **אינו** נחשב קבוע פשוט). למשל,

```
#define ATTR_LIGHT_BLUE ((ATTAR_BLUE)|(ATTAR_INTENSE))
```

- כשמאקרו מקבל פרמטרים, חובה להקיף כל פרמטר בסוגריים בכל פעם שיש שימוש בו. למשל,

```
#define SQUARE(x) ((x)*(x))
```

כתיבת ביטויים

- בתנאים מורכבים, יש להשתמש בסוגריים, גם אם אין צורך טכני בכך (גם אם הסוגריים לא משנים את סדר הקדימויות הרגיל). לדוגמה,

```
if ((a>b) || (a>c))
```

- אין להשתמש בערך של ביטוי השמה, חוץ מהמקרים הפשוטים ביותר, למשל:

```
if ((ptr = malloc(50) == null)
```

- יש להימנע משימוש בביטויים מותנים (?:) בתוך הקוד. עדיף להשתמש בפקודה if.
- מותר להשתמש בביטוי מותנה בתוך הגדרת מאקרו.

כתיבת פונקציות

הגדרת פונקציות

- פונקציות המשמשות לצרכים פנימיים של קובץ קוד יחיד, יוגדרו כסוג static.
- אם פונקציה אינה מחזירה ערך, יש להגדירה כ-void.
- אם פונקציה מחזירה int, יש לציין זאת במפורש, ולא לסמוך על ברירת המחדל.

- אם פונקציה לא מקבלת פרמטרים, יש לכתוב void בין הסוגריים בהכרזת הפונקציה (prototype). למשל,

```
void SoundBeep(void);
```

- בהכרזה על פונקציה המקבלת פרמטרים, יש לכתוב בהכרזת הפונקציה את סוג הפרמטרים, וגם שמות משמעותיים. למשל,

```
void SetArray(float * array , int size);
```

ערך המוחזר מפונקציה

- כשמשתמשים בפקודה return עם ערך, יש לכתוב את הערך בסוגריים.
- כאשר פונקציה מחזירה קוד המעיד על הצלחה או שגיאה, חובה לבדוק את הקוד ולהגיב בהתאם.
- כשיש אפשרות להתאושש ממצב השגיאה - יש לעשות זאת. לדוגמה,

```
while (scanf("%d" ,&value)!=1)
  {
     printf("Bad value... Try again: ");

     fflush(stdin);
  }
```

- כשאין אפשרות להתאושש ממצב שגיאה, יש לדווח על המצב במדויק ולצאת מהתוכנית. לדוגמה,

```
tree_root = malloc(NODE_SIZE);

if (tree_root == NULL)
  {
      printf("Out of memory... Can't continue.\n");
    exit(1);
  }
```

העברת פרמטרים לפונקציה

- **מבנים** (struct) או **איגודים** (union) המכילים מבנים יועברו לפונקציות בעזרת מצביע, **ולא** במלואם, כדי לחסוך שימוש ב-stack.
- כשפונקציה מקבלת מצביע לנתונים שאין בכוונתה לשנות (struct או מערך), היא תגדיר את הפרמטר כ-const. למשל,

```
float SumArray(const float * array, int count);
```

- כל סוג אחר של ערך יועבר כמצביע, רק אם הפונקציה צריכה לשנות משתנה של הפונקציה הקוראת. אין להעביר מצביעים שלא לצורך.

תיעוד קובץ קוד

כתיבה בהתאם לכללים שפורטו מאפשרת לכתוב קוד "**מובן מאליו**" (self documenting). יחד עם זאת, יש לזהות מקומות בקוד שאינם מובנים מאליהם, ולהוסיף הסבר בהערה. יש להשתדל לכתוב את ההסברים באנגלית ברורה, או בעברית ברורה - יש להימנע מכתיבת מילים עבריות באותיות אנגליות. גם רצוי להימנע מראשי תיבות שאינם מקובלים.

בראש כל קובץ יופיע בלוק הערה המתאר את הקובץ, תפקידו, הקשרים עם קבצים אחרים, ומעקב אחר היסטוריית השינויים בו. ראה קובץ file.txt.

- תיאור הקובץ (Description) יכלול סקירה של תפקיד הקובץ והפונקציות המסופקות לקבצים אחרים (פונקציות גלובליות).
- בהערות (Notes) יש לציין את האלמנטים החיצוניים בהם תלוי הקובץ (פונקציות ומשתנים גלובליים, וכל מידע חשוב אחר.

תיעוד פונקציה

בראש כל פונקציה יופיע בלוק הערה המתאר את הפונקציה, תפקידה, הקשרים עם פונקציות אחרות, ומעקב אחר היסטוריית השינויים בה. ראה קובץ file.txt.

- תיאור הפרמטרים יהיה קצר ויסביר את המשמעות של כל פרמטר. אין צורך לציין את שם הפרמטר או את הסוג שלו.
- תיאור ערך ההחזרה יהיה קצר ו/או יכיל פירוט של הערכים האפשריים.
- בסעיף "ערך ההחזרה" יפורטו גם שינויים המתבצעים במשתנים של הפונקציה הקוראת. למשל, בתיעוד הפנימי של פונקציה כמו ()gets, יש לכלול שורות אלו:

```
*  Return Value: - NULL ++> failed (EOF                    *
*            != NULL ++> ptr to buffer (parm 1)            *
*            -Caller's buffer (parm 1) contains line       *
```

- בהערות (Notes) יש לציין את האלמנטים החיצוניים בהם תלויה הפונקציה (פונקציות ומשתנים גלובליים), משתנים סטטיים בתוך הפונקציה, וכל מידע חשוב אחר.

```
/*   ---------------------------------------------------
 ;                                                     ;
 ;      File Name     : ...                            ;
 ;                                                     ;
 ;      Purpose       : ...                            ;
 ;                                                     ;
 ;      Returns       : ...                            ;
 ;                                                     ;
```

```
 ;     Notes          : ...                                   ;
 ;                                                            ;
 ;                                                            ;
 ;     Date          | Author      | Modification/Addition    ;
 ;     =============|=============|========================== 
 ;     ...          | ...         | Initial Version          ;
 ;     -------------|-------------|---------------------------
 ;     ...          | ...         | ...                      ;
 ;   ---------------------------------------------------------
*/
#include ...
     ...
     ...
/*   ---------------------------------------------------------
 ;     CONSTANTS
 ;   ---------------------------------------------------------
*/
#define ...
     ...
     ...
/*   ---------------------------------------------------------
 ;     TYPES
 ;   ---------------------------------------------------------
*/
     ...
     ...
/*   ---------------------------------------------------------
 ;     PROTOTYPES
 ;   ---------------------------------------------------------
*/
...    function(...);
     ...
     ...
/*   ---------------------------------------------------------
 ;     MAIN-PROGRAM
 ;   ---------------------------------------------------------
*/
int main(void)
{
     ...
     ...
}
/*   ---------------------------------------------------------
 ;     FUNCTIONS
 ;   ---------------------------------------------------------
*/
```

```
/*   ---------------------------------------------------
 ;                                                     ;
 ;     Procedure     : ...                             ;
 ;                                                     ;
 ;     Purpose       : ...                             ;
 ;                                                     ;
 ;     Parameters    : ...                             ;
 ;                                                     ;
 ;     Returns       : ...                             ;
 ;                                                     ;
 ;     Notes         : ...                             ;
 ;                                                     ;
 ;                                                     ;
 ;     Date          | Author     | Modification/Addition ;
 ;     ============================================    ;
 ;     ...           | ...        | Initial Version      ;
 ;     --------------------------------------------    ;
 ;     ...           | ...        | ...                  ;
 ;   ---------------------------------------------------
*/
...    function(...);
{
      ...
      ...
}
```

תיעוד של קטע קוד

לעיתים דרוש הסבר מפורט לקטע קוד כלשהו. לפני שורה או בלוק של קוד הדורשים הסבר, אפשר לכתוב בלוק הערה:

```
/*   ---------------------------------------------------
 ;   _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _    ;
 ;   _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _    ;
 ;   ---------------------------------------------------
*/
```

- גודל הבלוק יהיה לפי הצורך.
- ההזחה של הבלוק תהיה כמו זו של הקוד בו הוא נמצא.
- יש לתעד אלגוריתמים מורכבים, או החלטות שאינן מובנות מאליהן.

אם ההערה היא קצרה ומתייחסת לשורת קוד אחת, ניתן לכתוב אותה בסוף שורת הקוד, צמודה לימין העמוד. לדוגמה,

```
if (last_value<0)
           length-- ;              /* Discard bad value */
```

שימוש במשאבים

- משאבים יוקצו רק בעת הצורך, וישוחררו כשמסתיים השימוש בהם, גם במצב של יציאת חירום. במצבים קיצוניים אפשר להימנע מכך, כי המערכת עלולה לאבד את יציבותה, וכל פעולה נוספת עלולה להחמיר את המצב.
- אין לסמוך על שחרור אוטומטי של משאבים בסיום התוכנית, אלא צריך לשחרר אותם במפורש.
- רצוי שקטע קוד (פונקציה, קובץ, מודול) שביקש את הקצאת המשאב, יהיה גם האחראי לשחרורו.
- אין לפנות אל משאב לאחר ששוחרר (למשל, זיכרון).
- אין ל"שחרר" משאב שלא הוקצה, או שכבר שוחרר. יש לשים לב לכך בעיקר במהלך יציאת חירום ושחרור מערכות שהקצאתן לא הושלמה.

המשאבים שנמצאים בשימוש נפוץ:

- זיכרון (הקצאה דינמית)
- קבצים (פתיחה/סגירה)
- נעילות קבצים

הידור

- אם המהדר מפיק הודעות שגיאה, גם אם הן מסווגות כ**אזהרות** (warning) ולא כ**שגיאות** (error), חובה לתקן את הקוד.
- יש להשתמש באופציות של המהדר כדי שיפיק את מירב ההודעות.
- במקרים מיוחדים, מותר לבטל הפקת סוג הודעה זה או אחר, במיוחד כאשר ההודעות צפויות בגלל שיטת התכנות.
- יש להשתמש באופציות של המהדר, כדי שיאכוף כתיבה לפי תקן ANSI.

נספח ג'

פונקציות העכבר

תוכנות לעיבודים גרפיים, תוכנות שירות רבות וגם תוכנות מסחריות (ולמעשה, מי לא) משתמשות בעכבר כאמצעי קלט/פלט. נציג בפניך תריסר פונקציות המשמשות לתמיכה בעכבר ונסביר את תפקידן, השימוש בהן ודרך הפעלתן. ניתן גם הסבר על הארגומנטים הנמסרים (argument list) ועל ערך ההחזר (return value).

החומר בנספח זה נלקח באישור מתוך הספר:

C Programmer's Toolkit (2nd Ed.) / Jack Purdum / Que

המסך הווירטואלי של העכבר

לפני לימוד הפונקציות של העכבר, רצוי שתבין כיצד העכבר פועל בשילוב עם המסך. אפשר להשתמש במחשב במצבי מסך שונים. במצב **טקסט** מוגדר המסך בתור 80 טורים ו-25 שורות. במצב **גרפיקה** יכולה הרזולוציה להגיע ל-640 על 480 פיקסלים. בגלל הרזולוציות השונות משתמש העכבר במה שנקרא בשם **מסך וירטואלי** (Virtual Screen), המורכב ממטריצה של 640 נקודות אופקיות מול 200 נקודות אנכיות (זוהי הרזולוציה!). כל מצב מסך אפשרי ממופה לתוך מרחב קואורדינטות של 640x200, כשהקואורדינטה (0,0) משמשת נקודה שמאלית עליונה של המסך.

מצב-מסך ממשי (actual screen mode) קובע את צורת הגישה של העכבר אל המסך. לדוגמה, אם התוכנית שלך קיבלה מצב מסך לגרפיקה ברזולוציה של 640x200, תהיה התאמה של אחד לאחד בין המסך הווירטואלי למסך הממשי. פירוש הדבר, שתזוזה של נקודה אחת על המסך הממשי מתאימה לתזוזה של נקודה אחת על המסך הווירטואלי. ואולם, אם אתה משתמש במצב טקסט של 80x25 תהיה התאמה של שמונה לאחד בין המסך הווירטואלי למסך הממשי. לכן, תזוזה של יחידה אחת במסך המעשי נראית כמו תזוזה של **שמונה** יחידות במסך הווירטואלי.

נניח שקבעת את מצב המסך ל-40x25. במקרה זה תנועה של טור אחד במסך הממשי מקבילה לתנועה בת 16 יחידות במסך הווירטואלי. עליך לטפל בתרגום של ערכי המסך הממשי לערכי המסך הווירטואלי.

לאחדות מפונקציות העכבר שיידונו בפרק זה יש ארגומנטים המציינים את מיקום העכבר במונחים של שורות וטורים. ערכים אלה של שורות וטורים נקובים **תמיד** במונחים של המסך הווירטואלי. פירוש הדבר, שאם המסך נקבע למצב טקסט 80x25, יצוינו הקואורדינטות של העכבר במכפלות של שמונה בגלל ההתאמה של שמונה לאחד בין המסכים הווירטואלי והממשי. כדי לקבוע מהו המיקום (שורה וטור) במסך הממשי, עליך לחלק בשמונה את הקואורדינטות של המסך הווירטואלי.

תיווכח לדעת שהתכנות של העכבר יהיה הרבה יותר קל, אם תזכור שהעכבר "חושב" תמיד במונחים של מסך וירטואלי.

קלוט את סטטוס העכבר: mouse_status()

אם אתה רוצה להשתמש בעכבר, ההיגיון מחייב שתתחיל בבדיקה אם קיים עכבר במערכת. כל פונקציות העכבר משתמשות בפסיקה 0x33 ומכיון שפסיקה זאת היא במידת מה בלתי רגילה, רוב המהדרים אינם מספקים פונקציה מוגדרת מראש עבורה (שלא כמו בפסיקה 0x21). לכן עליך להשתמש בפונקצית הפסיקה הכללית int86(), המחייבת אותך להציג את מספר הפסיקה כאחד הארגומנטים של הפונקציה. הפונקציה הבודקת את הסטטוס של העכבר מוצגת בתדפיס הבא.

```
*****/
```

mouse_status

פונקציה זאת קובעת אם מותקן mouse driver וכמה לחצנים יש בו. מספר הפונקציה (הערך 0x00) נטען ב-ax, ומתבצעת קריאה לפסיקה 0x33.

```
Argument list:  int buttons    pointer to the number of
                               mouse buttons available.

Return value:   int            nonzero if the mouse is
                               installed; 0 if not. Note
                               that after the call to
                               int86(), the bx holds the
                               number of mouse buttons.
```

```
*****/

#include <dos.h>

int mouse_status(int *buttons)
{
   union REGS ireg;
```

```
   ireg.x.ax = 0x00;       /* Function 0x00--mouse status
*/
   int86(0x33, &ireg, &ireg);
                          /* bx is number of mouse buttons */
   *buttons = ireg.x.bx;
   return ireg.x.ax;
}
```

הפונקציה 0x00 מאפשרת לקבוע אם קיים עכבר במערכת. אחרי החזרה מקריאת הפונקציה ()int86, יהיו זמינות שתי יחידות מידע:

1. אם ireg.x.ax איננו אפס, פירוש הדבר שמותקן עכבר במערכת.
2. אם ireg.x.ax איננו אפס, אזי ireg.x.bx יכיל את מספר לחצני העכבר הזמינים.

נסכם זאת כך:

.x.ax = status	1 - עכבר מותקן.
	0 - אין עכבר מותקן.
ireg.x.bx	נותן את מספר הלחצנים של העכבר.

לרוב העכברים השכיחים יש שני לחצנים ויש עכברים עם שלושה לחצנים. השימוש בלחצנים מוסבר מאוחר יותר בפרק זה.

מכיון שאתה עשוי להשתמש מאוחר יותר בתוכנית במידע על מספר הלחצנים, יכיל המצביע אל "מונה הלחצנים" את התוכן של ireg.x.bx.

הדלק את סמן העכבר: ()move_cursor_on

הפונקציה ()mouse_status קבעה את עובדת קיומו של עכבר, ועתה עליך להפעיל (להדליק) את סמן העכבר.

```
*****/
```

move_cursor_on()

פונקציה זו מאפשרת להדליק את סמן העכבר. הפונקציה מחזיקה מונה פנימי הקובע אם הסמן דולק; אם המונה הוא 0, הרי שהסמן דולק. על ידי קריאה לפונקציה זו ולפונקציה 0x02 אתה יכול לכבות ולהדליק את סמן העכבר.

```
Argument list:  void
Return value:   void
```

```
*****/

#include <dos.h>

void mouse_cursor_on(void)
{
   union REGS ireg;

   ireg.x.ax = 0x01;                /* Toggle cursor on */
   int86(0x33, &ireg, &ireg);

}
```

קביעת המחדל של הפונקציה היא כיבוי הסמן של העכבר. בעת הדלקת הסמן של העבר הוא יופיע כברירת מחדל בצורת סמן-בלוק (block-style cursor). הפונקציה 0x01 של פסיקה 0x33 משמשת להדלקת סמן העכבר.

כבה את סמן העכבר: mouse_cursor_off()

סביר להניח שלא תרצה להחזיק את סמן העכבר דולק כל הזמן. אתה יכול לכבות אותו על ידי שימוש בקוד המוצג בתדפיס הבא.

```
*****/
```

mouse_cursor_off()

פונקציה זו מאפשרת לכבות את סמן העכבר על ידי הקטנה של המונה הפנימי שלו. עליך להפנות לסירוגין קריאות אל הפונקציה move_cursor_on() ואל move_cursor_off(), כדי שהפעולה תתבצע כהלכה.

```
Argument list:  void

Return value:   void
```

```
*****/

#include <dos.h>

void mouse_cursor_off(void)
{
   union REGS ireg;

   ireg.x.ax = 0x02;               /* Toggle cursor off */
   int86(0x33, &ireg, &ireg);
}
```

הפונקציה 0x02 של פסיקה 0x33 משמשת לכיבוי סמן העכבר. זכור שהעכבר פעיל גם כשהסמן כבוי. פירוש הדבר, שהפונקציות 0x01 ו0x02- אינן מכבות או מדליקות את **העכבר** עצמו. הן קובעות רק אם העכבר נראה לעין או לא.

קרא את המיקום והסטטוס של העכבר: read_mouse()

בתור התקן קלט, העכבר איננו מסוגל לומר הרבה למשתמש בו. ליתר דיוק, כל מה שאתה יכול להיודע מהעכבר הוא המקום שבו הוא ממוקם על המסך, ואם נלחץ אחד מלחצניו. הפונקציה read_mouse() תוכל לתת לך את שתי יחידות המידע האלו.

```
*****/
```

read_mouse()

פונקציה זו מאפשרת לקרוא את המיקום והסטטוס של העכבר. בעת החזרה מהפסיקה, יהיה זמין המידע הבא:

ireg.x.bx	סטטוס הלחצן, שבו סיביות 0 ו-1 מציינות את הלחצנים הימני והשמאלי. כאשר הלחצן לחוץ, ערך הסיבית הוא 1. אם יש לחצן אמצעי, הוא משתמש בסיבית 2.
ireg.x.cx	מצב אופקי של הסמן.
ireg.x.dx	מצב אנכי של הסמן.

מכאן, שערכי החזרה ב-ireg מספקים את המידע הדרוש.

```
Argument list:  int *row       pointer to the row position
                               of mouse cursor.

                int *col       pointer to the column
                               position of mouse cursor.

                int *button    pointer to return the button
                               status.

Return value:   void
```

```
*****/

#include <dos.h>

void read_mouse(int *row, int *col, int *button)
{
   union REGS ireg;

   ireg.x.ax = 0x03;   /* Read mouse position and status */
   int86(0x33, &ireg, &ireg);
```

```
   *button = iref.z.bx;
   *col = ireg.x.cx;
   *row = ireg.x.dx;

}
```

הארגומנט מועבר בשלושה מצביעים שלמים - אחד עבור כל מיקום בשורה ומיקום בטור, ואחד עבור סטטוס הלחצנים. בעת החזרה מהפונקציה 0x03, מכילים ireg.x.cx ו-ireg.x.dx את המיקום הנוכחי של העכבר לפי השורה והטור.

חשוב לציין שבעזרת הפונקציה ()read_mouse מתעדכן כל הזמן מיקום השורה והטור של העכבר. **המספר של הלחצנים אינו משפיע על המעקב אחר תנועות העכבר**. לכן, אתה יכול להשתמש ב-()read_mouse בכל פעם שאתה רוצה לדעת היכן העכבר נמצא, אך אינך מעוניין במספר הלחצנים.

ראינו שמספר הלחצנים מוחזר ב-ireg.x.bx. הסיבית 0 של ערך החזרה מתייחסת ללחצן השמאלי, סיבית 1 - ללחצן הימני וסיבית 2 - ללחצן האמצעי (אם ישנו). יהיה מן התועלת בכך שתגדיר את הסיביות האלו כקבועים סימבוליים לשימוש בתוכנית בצורה זו:

```
#define LBURRON    0x01
#define RBUTTON    0x02
#define MBUTTON    0x04
```

מכיון שהמידע בדבר מספר הלחצנים משמש ב-switch ובהודעות בקרה אחרות, מקילים הקבועים הסימבוליים על הבנת הקוד, מאשר אילו השתמשנו במיקומים של הסיביות.

קרא את לחצן העכבר: ()mouse_press

הפונקציה ()mouse_press מאפשרת למצוא את המיקום הנוכחי של העכבר על המסך. במקרים אחרים גם תהיה מעונין לדעת מתי נלחץ לאחרונה לחצן מסוים של העכבר. הפונקציה הניתנת בתדפיס קובעת את הסטטוס של העכבר בנסיבות כאלו.

```
/*****
```

mouse_press()

פונקציה זאת מאפשרת לבדוק איזה לחצן של העכבר נלחץ לאחרונה ומה היה המיקום (השורה והטור) של העכבר בעת הלחיצה. הלחצן שהמידע שלו מועבר לתוכנית הוא הלחצן הנבדק.

```
Argument list:  int row         the new row position.
                int col         the new column position.
                int button      button to check.
Return value:   int             the button status.
```

```
*****/

#include <dos.h>

int mouse_press(int *row, int *col, int *button)
{
   union REGS ireg;

   ireg.x.ax = 0x05;       /* Function to get mouse data */
   ireg.x.bx = button
                     /* Update only on this button press */
   int86(0x33, &ireg, &ireg);
   *col = ireg.x.cx;
   *row = ireg.x.dx;
   return (ireg.x.ax);          /* Return button status */
}
```

הפונקציה ()mouse_press שונה מהפונקציה ()mouse_read בבחינה אחת חשובה ביותר: **ערכי שורה-טור משתנים רק כאשר נלחץ הלחצן הקשור עם הארגומנט השלישי של הפונקציה (לחצן)**. פירוש הדבר, שהערכים שורה-טור אינם משתנים, אלא אם לחצנו על הלחצן המאוחסן ב-button. לדוגמה, נניח שקראת ל-()mouse_press, כאשר הארגומנט השלישי הוא MBUTTON. לדוגמה נניח גם, שלחצת על הלחצן האמצעי בשעה שהעכבר נמצא בשורה 50 ובטור 100. ערכי שורה-טור המוחזרים מהקריאה יהיו 50 עבור שורה ו-100 עבור טור. אם אתה מחליק את העכבר לאיזשהו מיקום אחר של שורה-טור, לא ישתנו ערכי השורה-טור, אלא אם נלחץ הלחצן האמצעי שנית. הפונקציה ()mouse_press מאפשרת לקרוא את מיקום שורה-טור בתנאי שלחצן מסוים אכן נלחץ.

קרא את שחרור לחצן העכבר: ()mouse_release

הפונקציה ()mouse_release מאפשרת לבדוק את המיקום שורה-טור, כאשר משחררים לחצן מסוים של העכבר.

```
*****/
```

mouse_release()

פונקציה זו מאפשרת לראות אם לחצן העכבר שוחרר. ערכי השורה והטור הם אלה שהיו קיימים בעת שהלחצן שוחרר.

אף על פי שהוא איננה כלול בפונקציה זו, המשתנה ireg.x.bx שומר את מספר השחרורים של הלחצן אחר הפסיקה. אחרי הקריאה לחצן זה מתאפס.

```
Argument list:  int row        the new row position.
                int col        the new column position.
                int button     the button to check.
Return value:   int            the button status: 1 if the
                               button is pressed, 0 if not.
```

```
*****/

#include <dos.h>

int mouse_release(int *row, int *col, int *button)
{
   union REGS ireg;

   ireg.x.ax = 0x06;       /* Function to get mouse data */
   ireg.x.bx = button          /* The button to inspect */
   int86(0x33, &ireg, &ireg);
   *col = ireg.x.cx;
   *row = ireg.x.dx;
   return (ireg.x.ax);          /* Return button status */
}
```

הפונקציה מתנהגת במידה רבה כמו ()mouse_press, אבל **העדכון של row ושל col מתבצע רק כאשר הלחצן משוחרר**, ולא בשעה שהוא נלחץ. לכן, אם המשתמש מעביר את העכבר בעוד button לחוץ, הערכים row ו-col יחזיקו את מיקום טור-שורה של העכבר בשעה שהלחצן משוחרר לבסוף. מידע זה יכול להיות חשוב במקרה שהמשתמש "גורר" דבר מה (לדוגמה, צלמית כלשהי) על פני המסך.

אם אתה רוצה לעקוב אחר מספר הלחיצות בלחצן, אתה יכול להעביר לפונקציה מצביע אחר. הפונקציה תכניס למצביע את הערך שנמצא ב-ireg.x.bx. המונה נמצא בתחום הערכים של שלם בעל-סימן (0 עד 32767) ומתאפס לאחר הקריאה.

קבע תחום אנכי לתנועת העכבר: mouse_vrange()

ביישומים מסוימים אתה עשוי להגביל את התחום שהעכבר יכול להשתמש בו. כשאתה כותב למשל, שורת תפריט עליונה (menu bar), סביר להניח שתרצה להגביל את תחום המסע האנכי של העכבר עד לשורה המשמשת לתפריט. התדפיס הבא מציג את הקוד המשמש לקביעת התחומים האנכיים לעכבר.

```
*****/
```

mouse_vrange()

פונקציה זאת מאפשר לקבוע ערכי מינימום ומקסימום לתחום מיקום השורה. התחום המותר הוא 0 עד 200.

```
Argument list:  int min        the min row position.
                int max        the max row position.
Return value:   void
```

```
*****/

#include <dos.h>

void mouse_vrange(int min, int max)
{
   union REGS ireg;

   ireg.x.ax = 0x08;   /* Function to set vertical range */
   ireg.x.cx = min;
   ireg.x.dx = max;
   int86(0x33, &ireg, &ireg);
}
```

שני הארגומנטים הניתנים ל-mouse_vrange הם ערכי מינימום ומקסימום של מיקום השורה שהעכבר יכול להשתמש בהן. הפונקציה 0x08 משמשת לקביעת הגבולות האנכיים. נזכירך, שערכי המינימום והמקסימום מתייחסים למסך הווירטואלי. לדוגמה, אם אתה כותב רק את הפונקציה

```
mouse vrange (0,7);
```

בשעה שהמסך במצב טקסט, העכבר יכול להשתמש בשורה הראשונה של המסך בלבד. ואולם שים לב לכך, שהקריאה לפונקציה בצורה זו

```
mouse-vrange (0,8)
```

הייתה מאפשרת לעכבר להשתמש בשתי השורות הראשונות של מסך הטקסט.

הגבולות האנכיים הם איפוא כוללניים מבחינה זו, שהתחום 0-7 מגביל את העכבר לשורה הראשונה של המסך הווירטואלי והתחום 0-8 כולל את השורות הראשונה והשנייה כאשר נמצאים במצב טקסט. דבר זה יכול להיראות מוזר, עד שתיזכר שהפינה השמאלית העליונה של המסך הווירטואלי היא במיקום (0,0). שמונה השורות הראשונות של המסך הווירטואלי משתמשות לכן בערכים 0-7 בהתאמה לשורה הראשונה (מס׳ 1) של מסך הטקסט הממשי. מכאן נובע גם, שמיקום אנכי בעל ערך 8 במסך הווירטואלי חייב להימצא בשורה השנייה של מסך הטקסט הממשי.

קבע תחום אופקי לתנועת העכבר: mouse_hrange()

את התחום האופקי של העכבר קובעים באותה דרך שבה נקבע התחום האנכי.

```
*****/
```

mouse_hrange()

פונקציה זו מאפשרת לקבוע ערכי מינימום ומקסימום עבור הקואורדינטות של הטור. התחום המותר הוא בטווח 0-640.

```
Argument list:  int min       the min column position.
                int max       the max column position.
Return value:   void
```

```
*****/

#include <dos.h>

void mouse_hrange(int min, int max)
{
   union REGS ireg;

   ireg.x.ax = 0x07; /* Function to set horizontal range */
   ireg.x.cx = min;
   ireg.x.dx = max;
   int86(0x33, &ireg, &ireg);
}
```

ערכי מקסימום ומינימום המועברים אל הפונקציה ניתנים במונחים של המסך הווירטואלי (כמו בפונקציות קודמות). דבר זה פירושו, שצריך לכפול בשמונה כל ערך המבוסס על המסך הממשי במצב טקסט, כדי שאפשר יהיה להשתמש בו בפונקציה זו. לדוגמה, הגבלת התנועה ל-40 הטורים הראשונים במצב טקסט פירושה שהערך של max המועבר לפונקציה יהיה 320. דבר זה נובע מהיחס 8:1 הקיים בין המסך הווירטואלי למסך הממשי.

קבע את מיקום העכבר: ()set_mouse

אתה יכול להציב את העכבר במיקום "שורה-טור" כלשהו, על ידי שימוש בפונקציה זו.

```
*****/
```

set_mouse()

פונקציה זו מאפשרת לך לקבוע את המיקום של העכבר. הערכים החדשים חייבים להימצא בתוך תחומי שורה-טור שנקבעו קודם. במצב טקסט, ערכים אלה מעוגלים לערכים הקרובים ביותר המותרים (בתוך התחומים).

```
Argument list:  int row        the new row position.
                int col        the new column position.
Return value:   void
```

```
*****/

#include <dos.h>

void set_mouse(int row, int col)
{
   union REGS ireg;

   ireg.x.ax = 0x04;   /* Function to set mouse position */
   ireg.x.cx = col;
   ireg.x.dx = row;
   int86(0x33, &ireg, &ireg);
}
```

הערכים row ו-col ניתנים במונחים של המסך הווירטואלי והם חייבים להימצא בתוך התחומים האופקיים והאנכיים, אם נקבעו. פונקציה זו יכולה להיות שימושית כאשר אתה רוצה להציב את העכבר באיזשהו מיקום מסוים (כגון בחירת מחדל של תפריט) כאשר התוכנית מתחילה.

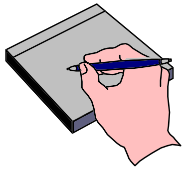
קבע סמן טקסט: ()mouse_cursor_t

הפונקציה מאפשרת להגדיר את סוג סמן העכבר שבשימוש, ויש להשתמש בה רק כאשר המסך נמצא במצב טקסט. הפונקציה מאפשרת להשתמש בשני סוגי סמנים: סמן מוגדר-חומרה או סמן מוגדר-תוכנה. נסביר כיצד אפשר לקבוע את הסמנים האלה, לאחר הצגת הקוד של הפונקציה.

*****/

mouse_cursor_t()

פונקציה זו מאפשרת להגדיר מחדש את סמן העכבר כאשר הוא נמצא במצב טקסט. שים לב לכך שהפונקציה פועלת רק במצב וידיאו טקסט (לא גרפיקה), ומכאן נובעת האות 't' בסוף שם הפונקציה.



הערה!

אם בוחרים בסמן-חומרה, הוא יפרש את הארגומנטים screen (מסך) ו-mask (מסכה) בתור שורות הסריקה ההתחלתית והסופית.

```
Argument list:  int type       the type of cursor:

                               0 =  software will define
                                    the cursor.

                               1 =  hardware will define
                                    it.

                int screen     defines the cursor screen.

                int mask       defines the cursor mask.

Return value:   void
```

```
*****/

#include <dos.h>

void mouse_cursor_t(int type, int screen, int mask)
{
   union REGS ireg;

   struct SREGS s;
                 / * CAUTION: MUST BE "SREG" FOR ECO-C88 */

   sergred(&s);                       /* Read segment reg  */
   s.es = s.ds;                       /* Load es with dseg */
   ireg.x.ax = 0x0a;                  /* Set text cursor   */
   ireg.x.bx = type;                  /* Type of cursor    */
   ireg.x.cx = (unsigned) screen;     /* Screen mask       */
   ireg.x.dx = (unsigned) mask;       /* Cursor mask       */
   int86(0x33, &ireg, &is);           /* Set all registers */
}
```

סמן מוגדר-חומרה הוא הסמן שנראה בדרך כלל בהנחייה של MS-DOS. אם רצונך להשתמש בסמן החומרה, אז הארגומנט הראשון (type) של הפונקציה mouse_cursor_t() ייקבע ל-1. סמן החומרה יכול להיות רק סמן מסוג "בלוק" ואפשר לקבוע את רוחבו בלבד. הרוחב מוגדר על ידי הארגומנטים השני והשלישי של הפונקציה. התחומים הם 0-7 עבור CGA display adepter ו-0-11 עבור Monochrome display adapter.

המשתנה screen משמש בתור שורת התחלה לסמן, והמשתנה mask משמש בתור שורת סיום. לכן, אם אתה משתמש ב-CGA, הקריאה הזו תגדיר "בלוק" מלא:

```
mouse_cursor_t(1, 0, 7);
```

מאידך, הקריאה הבאה תציג את הסמן בצורת קו-תחתי, המוכרת לנו יותר:

```
mouse_cursor_t(1, 6, 7);
```

כאשר משתמשים בסמן מוגדר-תוכנה, האפשרויות מגוונות יותר. כדי לבחור בסמן-תוכנה חייב הארגומנט הראשון של mouse_cursor_t() להיות 0 (אפס). הארגומנטים השני והשלישי מתפרשים אז כמסכות, שיש להחיל אותן על הנתונים שבמסך. המסכות הן ערכים של 16 סיביות, כפי שמוסבר ב**תרשים ג.1**. שמונה הסיביות הנמוכות מהוות את ייצוג התו על המסך. שים לב שגם המסכה של המסך וגם המסכה של הסמן הן בעלות אותו ייצוג.

משהוגדרו שתי המסכות, מבצעת הפונקציה mouse_cursor_t() קודם כל פעולת AND לוגית, עם נתוני התו שעל המסך (זכור שכל תו על המסך מייצג שני בתים בזיכרון: אחד עבור התו, ואחד עבור התכונות שלו). על התוצאה של פעולת AND זו מתבצעת פעולת OR בעזרת המסכה, כדי לקבוע את סמן הטקסט של העכבר.

סיביות:

15 7 0

B	back	i	fore	תו

תרשים ג.1:* *מסכות המסך והסמן

כאשר:

- B — אם סיבית זו היא 1, יעבור הסמן להבהוב; 0 - ללא הבהוב.
- back — קבע את דבר הרקע (background). הערכים 0 עד 7 קובעים את הצבע.
- i — אם סיבית זו היא 1, נקבל עוצמה (intensity) גבוהה; ואם 0 - העוצמה נמוכה.
- fare — קבע את צבע החזית (foreground). הערכים 0 עד 7 קובעים את הצבע.
- תו — ייצוג התו שעל המסך.

כדוגמה שכיחה נוכל לקבוע את מסכת המסך ל- 0x77ff ואת מסכת הסמן ל- 0x7700. הדבר מוצג ב**תרשים ג.2**.

מסך:

15				7 … 0
0	1 1 1	0	1 1 1	1 1 1 1 1 1 1

סמן:

15				7 … 0
0	1 1 1	0	1 1 1	0 0 0 0 0 0 0

תרשים ג.2:* *דוגמה של מסכות מסך וסמן

מסכות המסך והסמן המוצגות בתרשים 2 יוצרות סמן עכבר ההופך את צבעי **החזית** והרקע. מכיון שערכי המסך והמסכה של סמן העכבר המוצגים בתרשים 2 הם בשימוש כה שכיח, מומלץ ליצור עבורם הגדרות קבועות (#define) ולהשתמש בהם בתוכניות השונות.

```
#define  MCSCREEN  0x77ff        /* Mouse Cursor SCREEN */
#define  MCMASK    0x7700          /*Mouse Cursor MASK */
```

קבע סמן גרפיקה: mouse_cursor_g()

אם אתה משתמש במסך במצב גרפיקה, תוכל להיות גמיש יותר בהגדרת סמן העכבר. בתדפיס הבא מוצגת דוגמה של פונקציה העושה זאת.

```
*****/
```

mouse_cursor_g()

פונקציה זו מאפשרת להגדיר מחדש את סמן העכבר. הקוד המוצג כאן מגדיר את הסמן בצורת חץ. שים לב לכך שפונקציה זו פועלת רק במצב וידיאו גרפיקה (לא טקסט), ומכאן נובעת האות g שבסוף שם הפונקציה.

אם תדפיס את הסיביות הבינריות של המערך []mask, תוכל לראות כיצד מתבצעת הגדרת הסמן.

```
Argument list:  void

Return value:   void
```

```
*****/

#include <dos.h>

void mouse_cursor_g(void)
{
   static unsigned int mask[] = {
```

```
     0x3ff, 0x1ff, 0x0fff, 0x7ff, 0x3ff, 0x1ff, 0xff, 0x7f,
     0x3f, 0xif, 0x1ff, 0x10ff, 0x30ff, 0xf87f, 0xf87f, 0xfc3f
     0x00, 0x4000, 0x6000, 0x7000, 0x7800, 0x7c00, 0x7e00, 0x7f00,
     0x7f80, 0x78c0, 0x7c00, 0x4600, 0x0600, 0x0300, 0x300, 0x0180};
   union REGS ireg;
   struct SREGS s;
                / * CAUTION: MUST BE "SREG" FOR ECO-C88 */
   sergred(&s);                    /* Read segment reg    */
   s.es = s.ds;                    /* Load es with dseg   */
   ireg.x.ax = 0x09;               /* Set graphics cursor */
   ireg.x.bx = 0x01;               /* Column hot spot     */
   ireg.x.cx = 0x01;               /* Row hot spot        */
   ireg.x.dx = (unsigned) mask;    /* Pointer to mask     */
   int86x(0x33, &ireg, &s);        /* Set all registers   */
}
```

הפונקציה ()mouse_cursor_g המוצגת בתדפיס מוגדרת עבור סמן חץ. 16 הערכים הראשונים שבמערך []mask דומים במשמעותם לערכים המשמשים למסכת המסך. 16 הערכים האחרים מגדירים את מסכת הסמן. שתי מסכות אלו חייבות להימצא בזיכרון רציף, וכך הן הופכות למערך שלם אחד.

הקריאה ל-()segread דרושה כדי לאפשר לפונקציה למצוא את הנתונים המגדירים את סמן הגרפיקה. הכתובת של הקטע מועתקת לתוך ES לפני הקריאה, ו-ireg.x.dx מצביע על המסכה.

זכור שעליך לקרוא ל-()mouse_cursor_g לפני שתוכל לראות את סמן החץ, וכי הפונקציה פועלת רק במצב גרפיקה.

ספור את תנועות העכבר: ()mouse_counters

על ידי השימוש בפונקציה ()mouse_counters, אתה יכול לספור את תנועות העכבר. תנועות אנכיות או אופקיות מוחזקות ביחידת מדידה הנקראת בשם "מיקי" (mickey). כל יחידת מיקי מקבילה ל-1/100 האינץ' של תנועת העכבר. מכיון שאנו משתמשים ב-int בעל סימן, מוגבל התחום מ-32768- ועד 32767.

/*****

mouse_counters()

פונקציה זו מאפשרת לקרוא את המונים שורה-טור. כל יחידה מייצגת 0.001 אינץ', ויש לה תחום שלם בעל סימן (בערך פלוס/מינוס 32767).

שים לב לכך שהערכים המוחזרים מייצגים את השינוי במונים שחל מאז הקריאה האחרונה.

```
Argument list:  int *row       pointer to the row count.
                int *col       pointer to the column count.
Return value:   void
```

```
*****/

#include <dos.h>

void mouse_counters(int *row, int *col)
{
   union REGS ireg;

   ireg.x.ax = 0x0b;   /* Function to read motion counts */
   int86(0x33, &ireg, &ireg);
   *col = ireg.x.cx;
   *row = ireg.x.dx;
}
```

אל הפונקציה מועברים שני מצביעים שלמים בעלי-סימן. הפונקציה 0x0b של פסיקה 0x33 ממלאת במצביעים את הערכים המתאימים של המונים. ערך חיובי עבור col מציין תנועה לימין. ומכיון שהפינה השמאלית העליונה של המסך היא הכתובת (0,0), מציין ערך חיובי של row את תנועת העכבר כלפי מטה.

נספח ד'

תקציר פקודות ופונקציות (Quick Reference)

מבוא

בנספח זה כללנו את הפונקציות הבסיסיות בלבד, המהוות למעשה הרחבה לפקודות השפה.

לתיאור המבנה התחבירי השתמשנו בכללי הכתיבה הבאים:

< > תיאור כללי של פריט כלשהו.

דוגמה: <משתנה>, <טיפוס>, <ביטוי>, <ספריה> וכדומה.

[ ] ציון של איברי מערך.

[ ] איברים או ביטויים שהופעתם אפשרית, אך אינה הכרחית (אופציונלית).

דוגמה: <טיפוס> <שם משתנה> [= <ערך אתחול>] ;

כלומר, הגדרת משתנה יכולה להיעשות עם אתחול, או ללא אתחול.

[...] כאשר האיבר המופיע בין סוגריים מרובעים יכול להופיע מספר פעמים, הוספנו אחריו נקודות "...".

לדוגמה, כדי להגדיר רשימה בלתי מוגבלת של משתנים כתבנו:

<טיפוס> <שם משתנה> [, <שם משתנה>...] ;

ולמשל:

```
char c, ch, s, t ;
```

[[ ]] במקרים מסוימים השתמשנו בקינון של סוגריים מרובעים.

המשמעות: אם כוללים איבר הנמצא בתוך הסוגריים המרובעים החיצונים, ניתן עדיין להחליט אם לכלול במסגרת זו איבר שנמצא בסוגריים המרובעים הפנימיים.

דוגמה:

<טיפוס> <שם משתנה> [=<ערך אתחול>] [,<שם משתנה> [=<ערך אתחול>] [...]] ;

שימוש אפשרי: `char c='a', ch, s='\n', t ;`

הגדרות כלליות: טיפוסים, משתנים, קבועים, מצביעים, מערכים

טיפוסים בסיסיים	תו	`char`
	מספר שלם קצר (2 בתים)	`short`
	מספר שלם ארוך (2 בתים)	`int`
	מספר שלם (4 בתים)	`long`
	מספר ללא סימן	`unsigned`
	מספר עשרוני (נקודה צפה)	`float`
	מספר עשרוני (נקודה צפה בדיוק כפול)	`double`
	קבוע	`const`
	מבנה	`struct`
	איגוד	`union`
	הגדרת טיפוס חדש	`typedef`
	טיפוס לא מוגדר	`void`
קבוצות אחסון	אוטומטי	`auto`
	אוגר	`register`
	סטטי	`static`
	חיצוני	`extern`
קבוע תו	<תו> - כל תו הניתן להדפסה	`"<תו>"`
	הוראה לתחילת שורה חדשה	`'\n'`

	הוראת דילוג לתחילת שורה	'\r
	הוראת דילוג טבולטור	'\t
	הוראת היסט תו אחורה	'\b
	ציון של קו-נטוי	'\\'
	ציון גרש בודד	'\'
	הוראת קפיצת דף	'\f'
	ציון תו בעל ייצוג ASCII	'\<מספר אוקטלי>'
קבוע שלם	עשרוני	[-]<1-9>[<0-9>...]
	הקסדצימלי	0x<0-9,A-F,a-f>[<0-9>,A-F,a-f>...]
	אוקטלי	0<0-7>[<0-7...]
	שלם ארוך	[-]<1-9>[<0-9>...]L
קבוע נקודה צפה	רגיל	[-]<0-9>[<0-9>...][.[<0-9>...]]
	ייצוג מעריכי	[-]<0-9>[<0-9>...].[<0-9>...]E[<+,->]<0-9>[<0-9>]
קבוע מחרוזת		"<תו>[<תו>...]"
הערה		/* ...<תווים כלשהם>... */

שם משתנה שם משתנה מורכב מאחד עד שמונה תווים. התו הראשון חייב להיות אות מקבוצת התווים הבאה: a-z, A-Z או "_" (קו תחתון).

שאר התווים של השם יכולים להיות אותיות או ספרות: a-z, A-Z, "_", 0-9.

הגדרת משתנים

[<קבוצת אחסון>] <טיפוס> <שם משתנה> [=<ערך אתחול>]
[,<שם משתנה> [=<ערך אתחול>]...];

הגדרת מערך

<טיפוס> <שם מערך>[[<מספר איברים>]][={<רשימת ערכי אתחול>}];
<טיפוס> <הגדרת מערך>[,<הגדרת מערך>...];

הגדרת משתנה מצביע

```
<טיפוס>*<שם מצביע>[=<ערך אתחול>] [,*<שם מצביע>[=<ערך אתחול>]...];
```

הגדרת מערך מצביעים

```
<טיפוס>*<שם מצביע>[[<מספר איברים>]] [={<רשימת ערכי אתחול>}];
<טיפוס><הגדרת מצביע>[,<הגדרת מצביע>]...];
```

הגדרת מצביע לפונקציה

```
<טיפוס>*<שם מצביע>()[=<ערך אתחול>];
```

אופרטור, ביטוי, מבנה, איגוד

אופרטורים אונריים	מצביע	*
	כתובת של-	&
	קידום עצמי	++
	הקטנה עצמית	--
	גודלו של-	sizeof
	מספר שלילי	-
	שלילת ביטוי לוגי	!
	חילוף סיביות (NOT)	~
	קביעת טיפוס cast	(<טיפוס>)
אופרטורים זוגיים	חיבור	+
	חיסור	-
	כפל	*
	חילוק	/
	מודולו	%
	הזזת סיביות שמאלה (כפל)	<<
	הזזת סיביות ימינה (חילוק)	>>
	קטן מ- (לוגי)	<
	גדול מ- (לוגי)	>
	קטן או שווה ל- (לוגי)	<=

	גדול או שווה ל- (לוגי)	>=
	שווה (לוגי)	==
	לא-שווה (לוגי)	!=
	וגם (לוגי)	&&
	או (לוגי)	||
	XOR סיביות	^
	AND סיביות	&
	OR סיביות	|
אופרטורים להצבה מקוצרת	חיבור	+=
	חיסור	-=
	כפל	*=
	חילוק	/=
	מודולו	%=
	הזזת סיביות שמאלה	<<=
	הזזת סיביות ימינה	>>=
	AND סיביות	&=
	OR סיביות	|=
	XOR סיביות	^=
אופרטורים מיוחדים	ביטוי מותנה	? :
	שרשור ביטויים (פסיק)	,
ביטוי		<קבוע>
	או	<משתנה>
ביטוי מורכב		[<אופרטור אונרי>] <משתנה> [<אופרטור זוגי><ביטוי> . . .]
	או	<ביטוי> [<אופרטור זוגי><ביטוי> . . .]
	או	(<ביטוי>) ? <ביטוי> : <ביטוי>

מותר להשתמש בסוגריים בתוך הביטוי, כדי לשנות קדימות של שיערוך חלקי הביטוי.

מבנה

```
[<קבוצת אחסון>] struct [<שם טיפוס המבנה>] {
    ...<הגדרת תבנית, באמצעות טיפוסים מוכרים>...
} [<שם משתנה מבנה>] [={<רשימת אתחול>}]
    [, <שם משתנה מבנה> [={<רשימת אתחול>}] [...]];
```

חייב להופיע לפחות אחד מתוך: <שם טיפוס המבנה> או <תבנית>.

בהגדרת התבנית מוכרים כל הטיפוסים הבסיסיים וכל הטיפוסים שהוגדרו עד לנקודת הגדרת המבנה, כולל מבנים קודמים, איגודים וטיפוסים שנקבעו באמצעות typedef.

איגוד

```
[<קבוצת אחסון>] union [<שם טיפוס איגוד>] {
    ...<הגדרת תבנית>...
} [<שם משתנה איגוד>] [={<ערך אתחול>}]
    [, <שם משתנה איגוד> [=<ערך אתחול>] [...]];
```

החוקים הנוגעים למבנה תופסים גם לגבי איגוד.

הגדרת טיפוס חדש

```
typedef <טיפוס כלשהו> <שם סימבולי לטיפוס> [, <שם סימבולי>...];
```

ערך שמאלי ערך שמאלי הוא ביטוי סימבולי המתפרש ככתובת בזיכרון בו מצוי איבר, שבו ניתן לאחסן ערך. ביטוי זה יכול להיות:

משתנה פשוט	`<שם משתנה>`
מצביע	`*<שם משתנה>`
איבר במערך	`<שם משתנה>[<איבר>]`
איבר במערך מצביעים	`*<שם משתנה>[<איבר>]`

משפטים

משפט ריק	`;`
או	`<ביטוי>;`
או	`<משפט הצבה>`
או	`<משפט תנאי>`
או	`<משפט בקרה>`

הצהרה

משפט בודד:

```
<משפט בשפת C>
```

או משפט מורכב:

```
{
  <משפט בשפת C>
     [<משפט בשפת C>...]
}
```

משפט הצבה

```
<ערך שמאלי> = <ביטוי>;
```

או

```
<ערך שמאלי> [=<ערך שמאלי>...]=<ביטוי>;
```

או

```
<ערך שמאלי><אופרטור להצבה מקוצרת><ביטוי>;
```

תנאי

ביטוי כלשהו בעל המשמעות הבאה:

0	לא מתקיים
שונה מ-0	מתקיים

משפט if (תנאי)

```
if (<תנאי>)
        <הצהרה>
[else
        <הצהרה>]
```

משפטי בקרה

משפט while (לולאה)

כל עוד מתקיים <תנאי>

בצע את ה<הצהרה>

```
while (<תנאי>)
   <הצהרה>
```

משפט do..while (לולאה)

```
do
```

בצע <הצהרה>

```
   <הצהרה>
```

אם מתקיים התנאי בצע פעם נוספת

```
while (<תנאי>);
```

משפט for (לולאה)

```
for ([<ביטוי אתחול>];[<תנאי>];[<ביטוי עדכון>])
        <הצהרה>
```

בצע <ביטוי אתחול>. כל עוד מתקיים <תנאי> - בצע <הצהרה> ו-<ביטוי עדכון>.

משפט תווית

```
<תווית>:
```

משפט goto (דילוג)

דלג למשפט <תווית>.

```
goto <תווית>:
```

משפט break (יציאה מלולאה)

דלג קדימה, למשפט הראשון מחוץ למשפט הבקרה הנוכחי.

```
break;
```

משפט continue (התחל לולאה)

דלג לתחילת הלולאה הנוכחית.
בלולאת for בצע קודם <ביטוי עדכון>,
בלולאת do בצע קודם <תנאי>.

```
continue;
```

משפט switch (מיתוג)

```
switch (<ביטוי>) {
     [case <ערך שלם>:  [case <ערך שלם>:...]
                            [<משפט>...]
                                     ...]
        [default :
                              [<משפט>...] ]
```

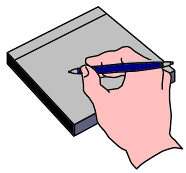
בצע שיערוך של <ביטוי>. מצא <ערך שלם> מתאים לתוצאת השערוך. בצע את המשפטים החל מנקודה זו ואילך. במידה ואין <ערך שלם> תואם, דלג ל-default (אם קיים) ובצע את המשפטים שאחריו.

פונקציות

רשימת פרמטרים פורמלים

```
[<שם פרמטר>[,<שם פרמטר>...] ]
```

כללי הכתיבה של <שם פרמטר> זהים לכללים של <שם משתנה>.



הערה!

ראה שם משתנה. שים לב לאפשרות של רשימה ריקה.

הגדרת פונקציה

```
[<טיפוס הפונקציה>]<שם הפונקציה> ( [<הגדרת טיפוסי פרמטרים ופרמטרים פורמלים>] )
     {
          [<הגדרת משתנים פנימיים>]

          [<משפט>]
     }
```

הביטוי <הגדרת טיפוסי פרמטרים ופרמטרים פורמלים> מכיל טיפוסים ופרמטרים אשר מקבלת הפונקציה.

<טיפוס הפונקציה> מגדיר חד-ערכית את טיפוס הערך המוחזר מן הפונקציה. אם אינו מופיע - ברירת המחדל הינה טיפוס int.

משפט return (חזרה מפונקציה)

return [([<ביטוי>])];

חזרה מפונקציה אל התוכנית הקוראת.

אם קיים <ביטוי> הוא ישוערך, יוסב לטיפוס הפונקציה ויהווה ערך מוחזר (לתוכנית הקוראת).

הצהרה על פונקציה

<טיפוס ערך מוחזר> <שם הפונקציה>();

קריאה לפונקציה

"קריאה" לפונקציה הינה הוראה להפעלה שלה. נגדיר פעולה זו באמצעות הרחבת המושג <ביטוי>.

ביטוי

<שם פונקציה> (<רשימת פרמטרים אקטואלים>)

(בנוסף למוגדר קודם לכן)

רשימת פרמטרים אקטואלים

[<שם משתנה>[,<שם משתנה>...]]

ערך <ביטוי> במקרה זה יהיה הערך המוחזר מהפונקציה. ניתן להשתמש בצורה זו של <ביטוי> בכל הגדרה בה מופיע <ביטוי> בהגדרות הקודמות.

קריאה לפונקציה באמצעות מצביע לפונקציה

(*<שם מצביע לפונקציה>) ([<רשימת פרמטרים אקטואלים>])

פונקציות ספריה

בגוף הספר הוצגו חלק מן הפונקציות הנלוות לכל מהדר של שפת C, אוסף הפונקציות שמוסברות בנספח זה אינו ממצה, כמובן, את כל האפשרויות שהספריות מעמידות לרשות המתכנת. יש פונקציות תקניות רבות נוספות שנמצאות בכל ספריה, ואין צורך לפרט את כולן בספר שמיועד ללימוד השפה. ראוי לציין, שבנוסף לספריות התקניות, כל יצרן של מהדר מוסיף פונקציות, כדי להקל על המתכנת במלאכתו, בתחומים שונים.

כדי להשתמש בפונקציות שברשימה זו, צריך המתכנת לכלול בראש התוכנית שורה, מן הסוג **#include <xxxxxx.h>** אשר בה, xxxxxx הוא שם של קובץ כתוב ב-C המסופק על ידי יצרן המהדר. קובץ זה מכיל **אבטיפוס** (prototype) של הפונקציה (תבנית). בהמשך נציין את שם הקובץ המתאים לכך קבוצת פונקציות. הסבר מפורט ראה בפרקים 5, 16... ובדוגמאות.

פונקציות הספריה מקובצות למשפחות, ולכן נציג את הפונקציות בחלוקה זו.

פונקציות קלט/פלט מהמסוף

```
char getch()
```

קריאת תו מהמקלדת מחזירה את התו שנקלט.

```
int putch(int ch);
```

כתיבת תו למסך.

```
int printf(const char *format [, argument, ...]);
int printf (<מחרוזת בקרה>[,<ביטוי>...])
```

פלט לפי תבנית. הדפסת מחרוזת הבקרה תוך שיבוץ הערכים של <ביטוי>.

```
int scanf(const char *format [, address, ...]);
int scanf (<מחרוזת בקרה>[,<כתובת של משתנה>...])
```

קליטה של נתונים לפי תבנית. הפונקציה מחזירה את מספר הנתונים שנקלטו.

```
char *strcpy(char *dest, const char *src);
```

העתקת המחרוזת המוצבעת על ידי s למחרוזת המוצבעת על ידי t. ההעתקה נמשכת עד תו אפס המסיים את מחרוזת s. תו האפס מועתק גם ל-t. הפונקציה מחזירה מצביע למחרוזת t (המועתקת).

פונקציות לטיפול במחרוזת

```
int strcmp(const char *s1, const char*s2);
```

השוואה לקסיקוגרפית בין שתי מחרוזות המוצבעות על ידי s1 ו-s2. הפונקציה מחזירה:

0	המחרוזות זהות
ערך חיובי	המחרוזת s1 גדולה מ-s2
ערך שלילי	המחרוזת s2 גדולה מ-s1

```
char *strcat(char *t, const char *s);
```

חיבור המחרוזת המוצבעת על ידי s למחרוזת המוצבעת על ידי t. יש לוודא ש-t מצביעה לאזור זיכרון גדול מספיק בכדי לקלוט את התוספת. הפונקציה מחזירה מצביע למחרוזת t (החדשה).

```
size_t strlen(const char *s);
```

אורך המחרוזת נמדד בבתים, עד ולא כולל, התו המסיים (תו אפס). הפונקציה מחזירה את אורך המחרוזת המוצבעת על ידי s.

פונקציות כלליות

```
double atof(const char *s);
```

הפיכת המחרוזת המוצבעת על ידי s למספר עשרוני עם נקודה צפה. המחרוזת s חייבת להכיל את התווים: "0"-"9", ".", "+", "-" ו-E בלבד. הפונקציה מחזירה את המספר העשרוני.

```
int atoi(const char *s);
```

הפיכת המחרוזת המוצבעת על ידי s למספר שלם. המחרוזת s חייבת להכיל את התווים: "0"-"9" ו-"-" בלבד. הפונקציה מחזירה את הערך השלם.

```
void *malloc(size_t size);
```

הקצאה דינמית של size בתים בזיכרון.
מחזירה את המצביע לאזור הזיכרון שהוקצה.

```
void exit(int status);
```

סיום של הרצת התוכנית וחזרה למערכת ההפעלה בקוד סיום status. סיום תקין הוא כאשר status=0.

פונקציות לטיפול בקבצים ללא מאגר

```
int creat(const char *path, int amode);
```

יצירת קובץ בשם המחרוזת המוצבעת על ידי filename. הערך access מגדיר את ההגבלות הכרוכות בגישה לקובץ. משמעותו של ערך זה משתנה בהתאם למערכת ההפעלה והמהדר. אם הקובץ כבר קיים, גודלו מכווץ ל-0.

מחזירה מספר ערוץ של הקובץ אם הכל כשורה, או (‎-1) אם הייתה תקלה.

```
int open(const char *path, int access [, unsigned mode]);
```

פתיחת קובץ בעל השם במחרוזת המוצבעת על ידי filename.
סוג הפתיחה נקבע על-פי ערך המחרוזת mode:

"0"	קריאה
"1"	כתיבה (מראשית הקובץ)
"2"	כתיבה (מהסוף והלאה)

מחזירה מספר ערוץ של הקובץ שנפתח אם הכל כשורה, או ‎-1 אם הייתה תקלה.

```
int close(int handle);
```

סגירה של הקובץ הפתוח על ערוץ handle.

מחזירה 0 אם הסגירה בוצעה כהלכה, או (-1) במקרה של תקלה.

```
int read(int handle, void *buf, unsigned len);
```

קריאה מקובץ בערוץ handle len בתים לתוך המחרוזת המוצבעת על ידי but. יש לוודא ש-buf מצביע למחרוזת בגודל מספיק לקליטת len בתים.

מחזירה את מספר הבתים שנקראו בפועל, או :

0	סוף הקובץ
(-1)	תקלה בקריאה

```
int write(int handle, void *buf, unsigned len);
```

כתיבה של len בתים מהמחרוזת המוצבעת על ידי buf לקובץ בערוץ handle. מחזירה את מספר הבתים שנקראו בפועל, או :

(-1) לציון תקלה כאשר נכתבים בפועל פחות בתים מ-bytes.

```
long lseek(int handle, long offset, int fromwhere);
```

הצבה של המצביע לקובץ בערוץ handle לבית בעל ההיסט offset מנקודת **fromwhere**.

"0"	מתחילת הקובץ
"1"	מנקודה נוכחית
"2"	מסוף הקובץ
offset	יכול להיות גם שלילי

מחזירה את המיקום הסופי ביחס לתחילת הקובץ.

פונקציות לטיפול בקבצים עם מאגר

```
FILE *fopen(const char *filename, const char *mode);
```

פתיחת קובץ בשם המצוי במחרוזת המוצבעת על ידי filename.
סוג הפתיחה נקבע על-פי ערך המחרוזת **mode** :

"r"	לקריאה

"w" לכתיבה (מראשית הקובץ)

"a" לכתיבה (מהסוף והלאה - הוספה)

מחזירה מצביע לקובץ שנפתח או 0 אם הייתה תקלה.

```
int fclose(FILE *stream);
```

סגירת ערוץ הקובץ stream. מחזירה את התו EOF אם הייתה תקלה, או 0, אם הקובץ נסגר כשורה.

```
int fgetc(FILE *stream);
```

קליטה של התו הבא מערוץ הקובץ stream, וקידום של המצביע לקובץ. מחזירה את התו הבא, או EOF, אם סוף הקובץ או הייתה תקלה.

```
int fputc(int c, FILE *stream);
```

כתיבה של c לקובץ stream, וקידום המצביע לקובץ. מחזירה את התו c אם הכל כשורה, או EOF במקרה של תקלה.

```
char *fgets(char *s, int n, FILE *stream);
```

כתיבה של השורה הבאה מהקובץ stream. השורה מסתיימת בתו '\n', או במידה ונקראו כבר n1 תווים. הפונקציה מוסיפה תו מסיים אפס (null) בסוף השורה בכל מקרה. מחזירה את המצביע ל-buf, או 0 במקרה של תקלה, או סוף הקובץ.

```
int fputs(const char *s, FILE *stream);
```

כתיבה של המחרוזת המוצבעת על ידי s לקובץ stream. הפונקציה איננה מוסיפה ביוזמתה את התו '\n' לסוף המחרוזת. היא מחזירה את התו האחרון שנכתב, או תו EOF במקרה שהתרחשה תקלה.

```
int fprintf(FILE *stream, const char *format
                            [, argument, ...]);
int fprintf(FILE *stream,<מחרוזת בקרה>[,<ביטוי>...])
```

פונקציה זו זהה ל-printf(), אלא שהפלט נכתב לקובץ file.

```
int fscanf(FILE *stream,  const char *format
                            [, address, ...]);
int fscanf(FILE *stream,<מחרוזת בקרה>[,<כתובת של משתנה>...])
```

פונקציה זו זהה ל-scanf(), אלא שהקלט מגיע מהקובץ file.

```
int fseek(FILE *stream, long offset, int origin);
```

הצבה של המצביע לערוץ קובץ stream, כך שיצביע לבית שמיקומו נמדד ב-offset בתים מנקודת ה-origin.

"0"	מתחילת הקובץ
"1"	מנקודה נוכחית בקובץ
"2"	מסוף הקובץ

offset יכול להיות גם שלילי

הפונקציה מחזירה 0 אם הביצוע נעשה כשורה או ערך שונה מ-0 במקרה שהתרחשה תקלה.

קדם מעבד

הגדרת <שם סימבולי>

```
#define<שם סימבולי>  <ערך מוחלף>
```

בכל מקום בתוכנית, בו מופיע <שם סימבולי> הוא יוחלף בביטוי <ערך מוחלף>. רק לאחר כל ההחלפות יתבצע תהליך ההידור.

```
#define <שם סימבולי>(<פרמטר פורמלי>[,<פרמטר פורמלי>...])<ערך מוחל>
```

הגדרת מאקרו

בכל מקום בתוכנית בו יופיע: (<רשימת פרמטרים אקטואלים>)<שם סימבולי> הוא יוחלף ב-<ערך מוחלף> תוך שיבוץ <פרמטרים אקטואלים> ב-<פרמטרים פורמלים> כפי שהם מופיעים ב-<ערך המוחלף>. רק לאחר כל ההחלפות יתבצע תהליך ההידור.

הידור מותנה

```
#if <תנאי> <רצף פקודות 1>
   [#else <רצף פקודות 2>]
#endif
```

כאשר מתקיים <תנאי> יבוצע הידור של <רצף פקודות 1>. אחרת - אם קיים else, יבוצע הידור של <רצף פקודות 2>.

הידור מותנה בהגדרת <שם סימבולי>

```
#ifdef <שם סימבולי> <רצף פקודות 1>
   [#else <רצף פקודות 2>]
#endif
```

אם הוגדר <שם סימבולי> באמצעות #define, יהודר <רצף פקודות 1>. אחרת, אם קיים else, יהודר <רצף פקודות 2>.

הידור מותנה באי-הגדרת <שם סימבולי>

```
#ifdef <שם סימבולי>  <רצף פקודות 1>
   [#else <רצף פקודות 2>]
#endif
```

זהה ל-#ifdef, אלא בהיפוך משמעות התנאי. כלומר, רק אם לא הוגדר <שם סימבולי> יבוצע הידור של <רצף פקודות 1>. אחרת, אם קיים else, יהודר <רצף פקודות 2>.

הפסקת הגדרה של שם סימבולי>

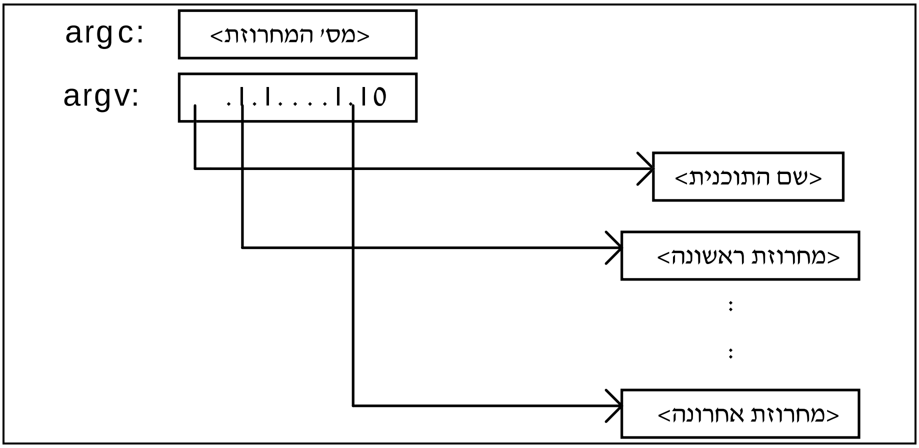
```
#undef <שם סימבולי>
```

מנקודה זו ואילך, <שם הסימבולי> אינו מוגדר לצורכי החלפה והידור מותנה.

רשימת פרמטרים לתוכנית הראשית

```
main (int argc, char *argvcj)
```

כאשר תופעל התוכנית ניתן להקיש בהמשך לשם התוכנית מחרוזות המופרדות בתו רווח. כאשר התוכנית תופעל יתקבל מבנה הנתונים הבא :



תרשים ד.1

נספח ה'

התקליטור המצורף וסביבת העבודה TCLite

תקליטור זה מופעל באופן אוטומטי עם הכנסתו לכונן.

אם התקליטור אינו מופעל באופן אוטומטי:

1. ודא שהתקליטור בכונן
2. בשולחן העבודה לחץ לחיצה כפולה על סמל **המחשב שלי**
3. לחץ לחיצה כפולה על סמל כונן התקליטורים
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**

התקליטור כולל:

- קטלוג ספרי מחשבים בהוצאת הוד-עמי
- **תוכנת TCLite** להרצת התוכניות שבספר
- **קוד מקור** של כל הדוגמאות שבספר
- תוכניות שירות מועילות
- שאלות נפוצות - לגבי רכישת ספרים בהוצאה
- אודות החברה
- דרכי התקשרות להוצאת הוד-עמי

הערה!
אם מנהל התקן כונן התקליטורים המותקן הוא 16 סיביות - ייתכן ותראה רק 8 תווים ראשונים של שם הקובץ (במקרה ובמקור הוא ארוך יותר). **הסיבה:** כונני תקליטורים במהירות x4 עובדים עם מנהל התקן שעבד בסביבת DOS ו-Windows 3.11 ויכול לעבוד גם עם Windows 9x, למעט היכולת לזהות קבצים עם שמות ארוכים. **הפתרון:** להתקין מנהל התקן 32 סיביות (אם קיים), או לקנות כונן תקליטורים חדש ולוודא שמצורף אליו מנהל התקן 32 סיביות.

קרא קובץ ONCD שנמצא בתקליטור לגבי תכולת התקליטור ולהוראות התקנה של חלק מהתוכנות המצורפות.

התיקיות הרלוונטיות לספר

התיקיות הרלוונטיות לספר זה הן:

Books\59159 שמכילה את קובצי קוד המקור הנמצאים בספר

Software\Tclite שמכילה את תוכנת TCLite (מהדר)

הפעלת התקליטור

במשך כל זמן ההפעלה חייב התקליטור להיות בכונן.

הוראות התקנה והפעלה מפורטות נוספות נמצאות בקובץ **ONCD.PDF** בתקליטור.

כדי לקרוא את תוכן הקובץ ONCD.PDF (הכולל פירוט מעוצב של תוכן התקליטור) עליך להתקין את תוכנת Adobe Acrobat Reader מהתפריט **התקנת תוכנות**.

בנוסף, תשמש אותך תוכנה זו לקריאה והדפסה של הפרקים לדוגמה, אותם תמצא בקטלוג ספרי המחשבים בהוצאת הוד-עמי.

התוכנה מאפשרת עיון והדפסת המסמך המוצג, כמו גם חיפוש בעברית ובאנגלית.

בעזרת תוכנה זו תוכל גם לקרוא את המסמכים אותם מפרסמת ההוצאה באתר האינטרנט.

קטלוג ספרים

בחלק זה תוכל לעיין בקטלוג הספרים בהוצאת הוד-עמי. עיון בקטלוג ספרי המחשבים של הוצאת הוד-עמי כולל:

- תוכן עניינים מלא של הספר
- פרק לדוגמה
- מגה-אינדקס לחלק מהספרים

את כל המידע ניתן לראות וגם להדפיס. לצורך זה יש להתקין את תוכנת Acrobat Reader מהתפריט **התקנת תוכנות**.

כדי לצפות בקטלוג העדכני ביותר ולצפות במבצעים המיוחדים מומלץ לבקר באתר האינטרנט של ההוצאה בכתובת www.hod-ami.co.il.

הקטלוג מומלץ לצפיה ברזולוציית מסך של 600x800.

הקטלוג מומלץ לצפייה באמצעות Internet Explorer 5 ומעלה. הדפדפן מצורף בתקליטור. כדי להתקין אותו בחר בתפריט **התקנת תוכנות** ובחר באפשרות **תוכנות נוספות**.

קוד המקור

בתקליטור נמצאים קובצי תוכניות. התוכניות נמצאות בשלוש ספריות נפרדות.

programs - התוכניות שנמצאות בספר ובדרך כלל מצוין שמם, Chp0001.c וכך הלאה, כאשר שתי הספרות הראשונות מציינות את מספר הפרק ושתי הספרות האחרונות את מספר התוכנית ברצף.

mesimot - התשובות לשאלות המוצגות בפרק 18. שמות הקבצים מצוינים בהתאם, Exr0401.c וכדומה.

targilim - תשובות לתרגילים המופיעים בששת הפרקים הראשונים. למשל, C1e1.c - פרק 1 תרגיל 1.

לכל ספר יש מספר ייחודי, הבנוי מה-דאנאקוד, המופיע על העטיפה האחורית של הספר, והוא מתחיל בספרות 59 (לדוגמה 59315).

מספר הפריט של ספר זה הוא : ***59159***

התקנת קוד המקור פירושה **העתקה** (לא התקנה) של הקבצים מהתקליטור לדיסק הקשיח במחשב באישי שלך.

לאחר שיועתק קוד המקור הוא יימצא בדיסק שבחרת, בתיקיה HodAmiBooks\59159.

> ***שים לב:*** *כאשר אתה עובד במהדר TCLite הכל מתבצע בסביבת המהדר. לא יוצרים קובץ EXE ולא יוצאים ל-DOS כדי להריץ אותו. הספר נכתב לשימוש במהדר רגיל ולכן יש לשים לב לשינויים כאשר עובדים ב-TCLite.*

תוכנת TCLite

תוכנת **TCLite** (Turbo C Lite), מהדר לשפת C של חברת Borland International, אשר מיועד למחשבי IBM PC ותואמים, הפועלים תחת מערכת הפעלה MS-DOS. זהו מהדר מהיר מאוד המסוגל, לפי דיווחי החברה, להדר מעל 7,000 שורות בדקה.

התוכנה כוללת:

- מהדר (compiler).
- עורך (editor) המשמש לכתיבת ותיקון התוכניות.
- מערכת תפריטים נוחה, לבחירת האופציות השונות במערכת.
- מקשר (linker).
- מערכת איתור שגיאות בהידור.
- מערכת לבניית תוכניות מקובצי מקור רבים.

אם מותקן כבר מהדר Borland במחשב, אל תתקין את TCLite אלא רק העתק את התוכניות מהתקליטור.

ההתקנה וההפעלה פשוטים.

הוראות התקנה והפעלה מלאות נמצאות בתקליטור בקובץ Software\Tclite\TCLiteGuide.pdf

התקנת התוכנה

התהליך הבא מעתיק את קבצי התוכנה לכונן הדיסק הקשיח :C שלך, לתיקיה בשם Tclite הנוצרת באופן אוטומטי.

אם התקליטור לא הופעל אוטומטית בצע שלבים 1-4. אם הופעל - עבור לשלב 5.

1. ודא שהתקליטור נמצא בכונן התקליטורים.
2. בשולחן העבודה לחץ לחיצה כפולה על הסמל **המחשב שלי**.
3. לחץ לחיצה כפולה על סמל כונן התקליטורים.
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**.
5. לחץ על **התקנת תוכנות**.
6. לחץ על לחצן האפשרויות ליד **אני מאשר את תנאי השימוש בתקליטור**.
7. לחץ **המשך**.
8. לחץ על **TCLite**.
9. התוכנה מותקנת בתיקיה **C:\tclite**.

בסיום שלב ההתקנה מועתק קובץ בשם c.bat לתיקיית השורש של כונן C. לנוחיותך ניתן ליצור **קיצור דרך** על שולחן העבודה (ראה ספרים בנושא Windows)

הפעלת התוכנה

1. לחץ לחיצה כפולה על קיצור הדרך ששמו TCLite על שולחן העבודה.
2. סביבת העבודה של TCLite מופיעה בחלון MS-DOS Prompt - TC, ובתוכה חלון (מסמך) בצבע כחול, ששמו: **NONAME00.CPP** (רק בפעם הראשונה שנכנסים לתוכנה).
3. כדאי לסגור את החלון **NONAME00.CPP** ע"י צמד המקשים <Alt>+<F3>, ומקבלים חלון אפור ריק, שבראשו רשימת תפריטים על רקע לבן. כרגע אתה מוכן להתחיל לעבוד בעזרת תוכניות הספר.

__שימו לב__: ברירת המחדל של ה- Compiler היא לקרוא ולשמור קובצי C/CPP. אתם כמובן קוראים קובצי C וכותבים קובצי C.

4. בחר בתפריט **Options** באפשרות **Environment** ולאחר מכן באפשרות **Editor**.
5. בתיבת הדו-שיח **Editor Options** בשדה **Default Extension** הקלד **C** במקום CPP. לחץ OK.
6. בחר בתפריט **Options** באפשרות **Save** כדי לשמור את השינויים שביצעת.

הרצת תוכניות

1. כדי לקבל תוכנית - יש להקיש <F3>. הקשה זו פותחת "חלון שיחה", שכותרתו הראשית: Load a File ("טען קובץ") וכותרת משנה ראשונה הינה: Name ואחריה, בתוך שורה כחולה עם הדגשה ירוקה כתוב: *.CPP . אם 5 תווים אלה (*.cpp) אכן מופיעים - הקש Backspace פעמיים כדי שיישאר רק c.*. אם אינך רואה דבר הקלד c.*, ולאחר מכן הקש <Enter>.
2. תעבור לחלון פנימי שכותרתו Files ("קבצים"), ועל רקע בצבע תכלת מצויים 2 טורים של שמות קבצים. באופן כללי, מקשי החיצים מאפשרים לטייל עם ה"סמן", שצבעו ירוק בהיר, לכל הקבצים.
3. למעשה, מה שמוצג כרגע על המסך הוא שמות הקבצים/תוכניות שבספריה programs. הקשה על <Enter> תכניס אותך לתוכנית. כדי לבחור ספריה אחרת (mesimot או targilim) הקש על \.. שמופיע בסוף הרשימה של התוכניות.
4. הקשת <Enter> על שם קובץ תפתח חלון כחול, שכותרתו במרכז הינה: שם הקובץ, ובתוך החלון יופיע הטקסט של תוכנית זו (או של קובץ זה).
5. כדי להריץ את התוכנית עליך להקיש על צמד המקשים: <Ctrl>+<F9>. המחשב יתרגם את התוכנית לשפת מכונה ויבצע אותה. כדי לראות את פלט התוכנית הקש <Alt>+<F5>. יש להקיש על מקש כלשהו כדי לחזור לחלון התוכנית.

6. המשך ההרצה של תוכניות נוספות מומלץ באופן הבא: תחילה לסגור את חלון התוכנית הנוכחית ע"י צמד המקשים <Alt>+<F3>. אח"כ ניתן לחזור על השלבים 1 עד 5, כאשר בשלב 3 אפשר לבחור תוכנית אחרת.

הערה: *בכל שלב בסעיפי הרצת התוכניות אפשר לבטל שלב כלשהו ולחזור לשלב הקודם ע"י המקש <Esc>.*

הערה חשובה!!!

אם המחשב שלכם מחובר לרשת, המלצתנו - בזמן אתחול המערכת להקיש F5 כדי לדלג על ביצוע הקבצים autoexec.bat ו-config.sys אם לא תעשו כך, ייתכן שבתוכניות תראו תווים לא מובנים במקום עברית.

הרצת תוכנית עם TCLite

1. כדי לקרוא לקובץ תוכנית קיימת יש לבחור בתפריט **File** באפשרות **Open**. נפתחת תיבת דו-שיח בשם: **Load a File**.
2. התיבה Name מכילה את הערך **C.*** (או **CPP.***, בהתאם למה שבחרת בתפריט Options, Environment, Editor). המשמעות היא שיוצגו בפניך כל התיקיות וכל הקבצים עם סיומת **C.** (או הסיומת **CPP.**)
3. אם קבצי התוכניות אינם נמצאים בכונן C, אלא בכונן **E**, לדוגמה, יהיה עליך להקליד בתיבה Name את רצף התווים הבא: **E:*.c** ולהקיש Enter.
4. בתיבה **Files** תוכל לראות את כל שמות התיקיות שבדיסק ואת שמות הקבצים לפי מה שרשום בתיבה **Name** (סיומת C או סיומת CPP). היעזר בפס הגלילה האופקי כדי לשוטט בתיבה Files.
5. כדי לעבור לתיקיה, סמן את התיקיה ולחץ **Open**.
6. כדי לחזור לתיקיה ממנה הגעת, סמן את רצף התווים **..** ולחץ **Open**. התו ..\\ נמצא בסוף רשימת הקבצים. אם יש הרבה קבצים באותה תיקיה לא כולם מופיעים בתיבה Files. השתמש בפס הגלילה האופקי כדי להגיע לסוף הרשימה.
7. כדי לפתוח קובץ, סמן את הקובץ ולחץ **Open**.
8. לאחר פתיחת קובץ תוכנית, תוכן הקובץ יופיע בחלון על רקע כחול שכותרתו היא שם הקובץ כולל נתיב התיקיות.
9. להרצת הקובץ יש לבחור בתפריט **Run** באפשרות **Run**.
10. כדי לראות את פלט התוכנית הקלד **Alt+F5** או בחר בתפריט **Window** באפשרות **Output**. היעזר בפסי הגלילה האנכי והאופקי של חלון Output כדי לראות את תוכנו.

11. חלון פעיל הוא חלון שהינך רואה ובפינתו השמאלית העליונה מופיע ריבוע [■]. כדי לסגור את החלון יש לנקוט באחת מהפעולות הבאות:

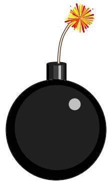
- בחירה בתפריט **Window** באפשרות **Close**.
- לחיצה אחת באמצעות העכבר על הריבוע.

12. מעבר בין חלונות (Message, Output, Watch, User Screen) באמצעות תפריט **Window**.
13. ליצירת קובץ תוכנית חדש יש לבחור בתפריט **File** באפשרות **New**.
14. לשמירת קובץ יש לבחור בתפריט **File** באפשרות **Save**.
15. ליציאה ממהדר Tclite בחר בתפריט **File** באפשרות **Quit**.
16. סגור את החלון על ידי לחיצה על ☒.

קרא קובץ Software\TCLite\TCLiteGuide.pdf שנמצא בתקליטור ללימוד עקרונות העבודה ב-TCLite.

מה עוד בתקליטור?

הוצאת **הוד-עמי** מפיצה תוכנות אלו כבונוס ללקוחות ההוצאה, ואינה מתיימרת לגבות תשלום עבור התוכניות המצורפות ו/או לתמוך בהם.



שים לב!

השימוש בתקליטור זה הוא על אחריותו הבלעדית של המשתמש. המוצרים המותקנים בתקליטור זה מסופקים באחריות החברות המייצרות אותם. הוצאת **הוד-עמי** אינה אחראית, בכל צורה שהיא, לאופן ולטיב התוכנות המותקנות.

בכל שאלה לגבי תוכנה הנמצאת בתקליטור, יש לפנות למפתחי התוכנה (כל תוכנה בנפרד) כפי שמצוין בקובצי העזרה של התוכנה המדוברת.

הקבצים הם גרסאות ShareWare ו-FreeWare.

גרסת ShareWare מאפשרת לך, המשתמש, לבדוק את יעילות התוכנה ואת תאימותה לעבודה אותה אתה מבצע. אם נמצאה התוכנה מתאימה לצרכיך, עליך לשלם למפתחיה תשלום סמלי (לפי הרשום בקובצי העזרה של כל תוכנה ותוכנה בנפרד) כדי לקבל רישיון מלא לשימוש בה. קבלת רישיון לשימוש בתוכנה יפתח בפניך מיגוון אפשרויות שלא עמדו לרשותך בהפעלת גרסת ה-ShareWare.

הוצאת הוד-עמי אינה תומכת בתוכנות המסופקות בתקליטור זה.

קרא על התוכנות המצורפות בתקליטור בקובץ ONCD.PDF.

התוכנות המומלצות להתקנה

1. תוכנת Adobe Acrobat Reader כדי שניתן יהיה לעיין וגם להדפיס את הפרקים לדוגמה ואת תוכן העניינים של כל ספר בהוצאת הוד-עמי.
2. תוכנת TCLite.

להתקנת שאר התוכנות יש ללחוץ על **התקנת תוכנות**, לאשר את תנאי השימוש בתקליטור וללחוץ על **תוכנות נוספות**. פירוט רחב יותר לגבי חלק מהתוכנות המצורפות בתקליטור תוכל לקרוא בקובץ ONCD.PDF אשר בתקליטור.

Adobe Acrobat Reader

1. ודא שהתקליטור נמצא בכונן התקליטורים
2. בשולחן העבודה, לחץ לחיצה כפולה על הסמל **המחשב שלי**
3. לחץ לחיצה כפולה על סמל כונן התקליטורים
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**
5. לחץ על **התקנת תוכנות**
6. לחץ על לחצן האפשרויות ליד **אני מאשר את תנאי השימוש בתקליטור**
7. לחץ **המשך**
8. לחץ על Acrobat Reader
9. אשף ההתקנה מתקין את הרכיבים הנדרשים.

   במהלך ההתקנה עליך ללחוץ על **Next, Accept** ו-**Next** פעם נוספת כדי לבצע את ההתקנה.
10. בסיום ההתקנה עשויה להופיע על המסך תיבת דו-שיח **התנגשות בין גירסאות** ומייד אחר כך להיעלם. על המסך תופיע תיבת הודעה של תוכנית ההתקנה. לחץ על **אישור.** בתיבת הדו-שיח **התנגשות בין גירסאות** לחץ על **כן**, כדי לשמור את גרסת הקובץ שלך.

שאלות נפוצות

בחלק זה של התקליטור תמצא אוסף של שאלות נפוצות ותשובות במיגוון נושאים: איך מחפשים בספר? כיצד מזמינים ספר דרך האתר באינטרנט? למי פונים בבקשת תמיכה? האם ניתן לבצע הזמנה טלפונית? וכדומה.

הערה חשובה!

הוצאת הוד-עמי אינה תומכת בתוכנות המסופקות בתקליטור זה. התוכנות ניתנות ללקוח כבונוס, לשימושו האישי, והן מסופקות AS-IS כפי שסופקו להוצאת הוד-עמי על ידי יצרן התוכנה.

התמיכה ניתנת ללקוחות ההוצאה בנוגע לספרים הרואים אור בהוצאה.

ניתן לשאול שאלות, להציע הצעות ולבקש עזרה בהפעלת קוד או קובץ המוזכר בספר.

מחלקת התמיכה של הוצאת הוד-עמי אינה תומכת ביישומים ובתוכנות המסופקות בתקליטור.

צור קשר

תוכל ליצור עמנו קשר באמצעי התקשורת הבאים:

טלפון: 09-9564716

פקס: 09-9571582

דואר אלקטרוני בנושא כללי: info@hod-ami.co.il

דואר אלקטרוני בנושאי תמיכה: support@hod-ami.co.il

אנשי התמיכה שלנו ערוכים לתת לך את העזרה הדרושה לך בשימוש בספר.

אין אנו נותנים תמיכה הקשורה בתוכנות המצורפות בתקליטור.

בפנייתך אלינו לקבלת תמיכה, אנא הצטייד בנתונים הבאים:

1. שמו המלא של הספר
2. מספר דאנאקוד, כפי שמופיע על הכריכה האחורית של הספר (למשל בספר זה 259-10159)
3. פרטים מדויקים של הבעיה (הכוללים מספר פרק, מספר עמוד, שם קובץ)
4. אמצעי קשר בו נוכל להשיג אותך: טלפון נייד, פקס, E-Mail.

אם יש לך שאלה או הערה בנוגע לספר מספרי הוצאת הוד-עמי, נשמח לשמוע ממך. יש מספר דרכים בהן תוכל להעביר לנו את הערותיך:

בדואר אלקטרוני: support@hod-ami.co.il

בפקס: 09-9571582

בדואר: ת.ד. 6108 הרצליה 46160

בפנייתך אנא ציין את הפרטים הבאים: שמו המדויק של הספר, מספר עמוד, מספר שורה מלמעלה/מלמטה, מה לא בסדר, מה לדעתך צריך להיות.

תודה רבה שבחרתם הוד-עמי

טלפון: 09-9564716 פקס: 09-9571582

אתר באינטרנט: http://www.hod-ami.co.il

דואר אלקטרוני: info@hod-ami.co.il

פנה לקובץ ONCD.PDF לפרטים נוספים על תכולת התקליטור.

אינדקס עברי

ב

ג

ד

ה

ז

ל

מ

נ

ס

ע

ק

אינדקס לועזי

B

C

D

E

N

O

P

Q

R

S

ת.ד. 6108 הרצליה 46160 **הזמנות באתר**

info@hod-ami.co.il www.**hod-ami**.co.il

הוצאת הוד-עמי ספרי מחשבים

קטלוג ספטמבר 2015

	כולל	עמ'	מחיר*
אינטרנט - מפתחי אתרים/גרפיקה			
אמא, אבא - בניתי אתר באינטרנט (HTML)		256	29
הגדל את הכנסות העסק שלך באמצעות **פרסום בגוגל Google AdWords**		300	249
מבוא לתכנות בסביבת אינטרנט – הכל כלול בספר אחד HTML + JS + ASP		192	9
מבוא לתכנות בסביבת אינטרנט - מבוא ו- **HTML** - חלק 1 מתוך 3 - מהד' 3		320	59
מבוא לתכנות בסביבת אינטרנט - **JavaScript** - חלק 2 מתוך 3		240	49
מבוא לתכנות בסביבת אינטרנט - **ASP** - חלק 3 מתוך 3 - מהד' 3		192	49
HTML5 המדריך לבניית אתרים ולמערכות WEB, הדור הבא – מהד' 2 חדש		384	159
The **Java** Tutorial סדנת לימוד	CD	768	179
JavaScript סדנת לימוד		586	159
מדריך **ASP.NET MVC 4**		514	199
ASP.NET 3.5 סדנת לימוד בשפות #C ו-VB		824	99
תכנות			
Code Complete – מדריך מעשי לפיתוח תוכנה		288	139
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
Visual C# 3.0 סדנת לימוד		656	99
ללמוד **C** - מהד' 3		480	139
שפת **אסמבלי** למחשב האישי, מהד' 2		314	89
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 4		152	95
PC - חומרה, תוכנה ורשתות			
מדריך Hacking ואבטחת מידע, מהד' 2		428	169
מדריך **חומרה ותוכנה** לטכנאי PC - מהד' 5	CD	752	189
Windows 7/8 נושאים מתקדמים		140	49
מדריך **רשתות** לטכנאי PC ולמנהלי רשת - מהד' 4		608	219
Windows			
Windows 8.1 מדריך למשתמש חדש		544	149
Windows 8 מדריך למשתמש		438	19
Windows 7 צעד-אחר-צעד		272	19

* מחיר מומלץ לצרכן כולל מע"מ

היכנס לאתר להתעדכן בספרים החדשים ומבצעים

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il

	כולל	עמ'	מחיר*
גרפיקה			
Flash – ספר הדרכה ותרגילים		122	64
אינדיזיין – ספר הדרכה ותרגילים		132	72
Illusatrator – ספר הדרכה ותרגילים		120	64
Photoshop צעד אחר צעד (צבע מלא, למתחילים), מהד' 3		200	159
מדריך לתוכנת העיצוב והאנימציה **3ds max** (2 כרכים)	CD	1400	289
OFFICE			
יישומי סטטיסטיקה בגיליון אלקטרוני Excel חדש		86	69
טבלאות ציר – ניתוח נתונים חכם		116	87
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 4		152	95
Word 2010 צעד אחר צעד		160	69
Excel 2010 סדנת לימוד		344	129
PowerPoint 2010 צעד אחר צעד		202	69
Outlook 2010 צעד אחר צעד		190	69
Access 2010 סדנת לימוד		360	179
עוד ספרים בגרסאות קודמות (2007 ו-2003) ניתן למצוא באתר הוד-עמי			
ניהול, כלכלה ושונות			
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
ניהול ממוקד לעשות יותר עם מה שיש (כריכה קשה) - מהד' 4		358	133
לי זה עולה יותר (תמחיר) (כריכה קשה) - מהד' 3		368	129
מערכות מידע			
Oracle SQL יכולות מתקדמות		626	249
SAS (**Statistical Analysis System**) – ספר לימוד		256	169
בסיסי נתונים ושפת **SQL** – עקרונות ועיצוב		648	149
ניתוח מערכות מידע כולל מתודולוגיית ה- **UML**		818	229
המדריך העברי השלם **UML**		346	329
ספרים דיגיטליים			
לרכישת ספרים לצפייה בפורמט PDF היכנס לאתר לקטגוריה "ספרים דיגיטליים"			
קבצי תרגול לספרים			
קבצי תרגול לספרים שונים תמצא באתר בקטגוריה "קבצי תרגול לספרים"			

* מחיר מומלץ לצרכן כולל מע"מ. קטלוג 9/2015

היכנס לאתר להתעדכן בספרים החדשים ומבצעים

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il